Titre: Title:	Modélisation et conception d'une plate-forme de traitement et transmission de signaux vidéo numériques
Auteur: Author:	M. Dubois
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dubois, M. (2004). Modélisation et conception d'une plate-forme de traitement et transmission de signaux vidéo numériques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7479/

NOTE TO USERS

UNIVERSITÉ DE MONTRÉAL

MODÉLISATION ET CONCEPTION D'UNE PLATE-FORME DE TRAITEMENT ET TRANSMISSION DE SIGNAUX VIDÉO NUMÉRIQUES

MATHIEU DUBOIS

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE EN SCIENCES APPLIQUÉES
EN GÉNIE ÉLECTRIQUE
JUILLET 2004

Your file Votre référence
ISBN: 0-612-97942-3
Our file Notre référence
ISBN: 0-612-97942-3

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Le titre du mémoire :

MODÉLISATION ET CONCEPTION D'UNE PLATE-FORME DE TRAITEMENT ET TRANSMISSION DE SIGNAUX VIDÉO NUMÉRIQUES

Présenté par : Mathieu Dubois

En vue de l'obtention du diplôme de : Maîtrise en sciences appliquées

A été dûment accepté par le jury de l'examen constitué de :

M. Yves Audet, Président

M. Yvon SAVARIA, Ph.D., membre et directeur de recherche

M. Guy BOIS, Ph.D., membre et codirecteur de recherche

M. Mohamad Sawan, membre

REMERCIEMENTS

J'aimerais remercier tout d'abord mon directeur de recherche, Yvon Savaria et la compagnie Gennum pour leur soutien financier. De plus, je remercie messieurs Savaria et Guy Bois pour leur encadrement pendant l'élaboration de ce projet de recherche.

Je tiens à remercier tous les étudiants du GRM (Groupe de Recherche en Microélectronique) qui ont contribué à l'avancement du projet de conception sur les convertisseurs de protocoles réseaux et de la plate-forme de traitement vidéo. Plus particulièrement, j'aimerais remercier Serge Catudal pour le développement en C++ de l'architecture de Wiener. En plus, je remercie Maurice Kouam pour avoir fourni un module de réducteur de bruit en SystemC pour le développement de ma méthode. Ces personnes et Robert Groulx ont collaboré à la nouvelle plate-forme de traitement vidéo.

Enfin, je remercie mes parents et mon frère Martin Dubois pour leur soutien dans ce travail.

RÉSUMÉ

L'application visée dans ce projet de maîtrise concerne une plate-forme SoC (System on Chip) pour le traitement et la transmission vidéo. La transmission sera réalisée par un processeur réseau d'abord conçu pour la conversion de protocoles. Le traitement vidéo sera exploré au travers une plate-forme vidéo conçue pour exécuter l'algorithme de Wiener. Dans les deux cas, nous emploierons un modèle de plate-forme générique.

Ce travail discutera d'une nouvelle méthode pour concevoir un SoC avec plusieurs langages et il discutera des modules d'intercommunication dans une plate-forme SoC. La réduction du temps, des coûts de développement et une mise en marché plus rapide découleront de cette recherche. Une exploration des capacités de quelques langages permettra d'élaborer une nouvelle méthode à double profilage utilisant le meilleur de chacun.

Relier l'ensemble des modules d'une plate-forme SoC exige la compatibilité des modules. Plusieurs solutions sont disponibles, mais elles sont caractérisées par une bande passante insuffisante ou un manque de flexibilité. Ce travail propose une architecture d'intercommunication pour un milieu de communication flexible de haute performance pouvant fournir une largeur de bande variable. Il est basé sur le bus AHB de AMBA. L'architecture proposée a été implémentée dans l'environnement HDL Designer Series™ de Synopsis, en utilisant la technologie CMOS 0.18 micron avec des outils de Cadence pour valider le concept proposé. Nos travaux montrent qu'il est possible d'obtenir un pont AHB/AHB multi-fréquentiel avec de nouvelles caractéristiques. Ce pont permet la synchronisation automatique entre les domaines d'horloges opérant autour de 500MHz.

ABSTRACT

The thesis is concerned will the development of a System on Chip (SoC) platform for transmission and processing of video. The transmission will be done using a network processor first developed for protocol conversion applications while video processing will be explored through a video platform developed for the Wiener noise reduction algorithm. In both cases, we will use a generic platform model.

This work will discuss a new method to design a SoC using several languages, and it will explore means of interconnecting modules in a SoC platform. The research aims at reducing development time and costs. We explore several languages that can be leveraged in an effective SoC platform design methodology. We propose a method based on a double profiling to leverage advantages of the various languages.

Interconnecting modules in a SoC platform requires modules compatibility. Several solutions are available, but they either lack the necessary throughput or flexibility. This work proposes an interconnection architecture to implement a flexible on-chip high-performance communication medium that can provide variable bandwidth. It is based on the AHB AMBA bus. The proposed architecture has been implemented with the Synopsis HDL Designer environment and laid out using a 0.18 micron CMOS with Cadence tools to validate the proposed concept.

Our results demonstrate that it is possible to get a multi frequency AHB/AHB bridge with several interesting features that can be used to synchronize multiple clock domains with different frequencies operating up to 500MHz.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES NOTATIONS ET SYMBOLES

AC : Address Converter
AHB : Advanced High Performance Bus
AMBA : Advanced Microcontroller Bus Architecture
APB : Advanced Peripheral Bus
ASB : Advanced Serial Bus
ASIC : Application Specific Integrated Circuit
CG : Checksum Generator
CMOS : Complementary Metal-Oxide Semiconductor
CV : Checksum Verifier
DSP : Digital Signal Processor/Processing
FPGA : Field Programmable Gate Array
GF : General Formatter
GRM : Groupe de recherche en microélectronique
IP : Intellectual Property
OSI : Open System Inteconnect
MB : Mailbox
MMAD : Main Memomry Address Distributor
PAx : Packet Assembler (instance #x)
PRx : Packet Receiver (instance #x)
RAM : Random Access Memory
RISC : Reduced Instruction Set Computer
SoC : System-On-Chip
SRAM : Synchronous Random Access Memory
VHDL : Very high speed integrated circuit Hardware Description Language

LISTE DES ANNEXES

AVANT-PROPOS

Cette recherche a été effectuée en étroite collaboration avec deux groupes de la Société Gennum, à Burlington, Ontario. Le premier groupe s'intéresse au développement d'un système de transmission de données vidéo à travers des réseaux de communications hétérogènes et le deuxième s'intéresse au développement d'une plate-forme de traitement vidéo.

Cette recherche a visé deux buts principaux, soit une modélisation hétérogène et la conception d'un bus de haute performance. La modélisation hétérogène permet l'utilisation efficace de plusieurs outils de conception communicant entre eux. Nous modéliserons une plate-forme SoC spécialisée en transmission et en traitement vidéo. Un autre aspect important se rapporte aux interconnexions entres les modules de la plate-forme. Les systèmes d'interconnexions des SoC manquent soit de bande passante ou de flexibilité. Dans le cadre de cette recherche, nous élaborerons des mécanismes permettant de réaliser des interconnexions flexibles et de haute performance.

CHAPITRE 1
INTRODUCTION

1.1 Motivations

Les systèmes embarqués demandent de plus en plus de capacité de calcul. Par exemple, une application vidéo HTDV exige une plate-forme capable de traiter 1.49 Gbps [45]. Les systèmes doivent être configurable et de haute performance. Aussi, le temps de mise en marché d'un produit doit être le plus court possible. Il est nécessaire d'avoir une méthode de conception descendante qui va du haut niveau vers le bas niveau. Donc, le premier objectif de ce mémoire sera l'exploration d'un environnement hétérogène permettant l'utilisation appropriée des langages de modélisation selon leurs capacités. De plus, les systèmes possèdent des interconnexions entre les modules matériels ou logiciels pouvant mener à des latences significatives. Il y a une division des modules en sous système tel que les processeurs, la mémoire et les périphériques d'entrées et de sorties. Il existe des protocoles de communication tel qu' AMBA, CoreConnect, etc. Certains d'entre eux manquent de flexibilité ou offrent une bande passante insuffisante pour les besoins de l'application.

Le deuxième objectif est de concevoir une plate-forme SoC générique pour la transmission et le traitement vidéo. Cette plate-forme pour la transmission est réalisée dans le cadre du projet de recherche Netpro du GRM (Groupe de recherche en microélectronique) de l'École Polytechnique. Il vise à l'élaboration d'un convertisseur de protocoles. Quant à elle, la plate forme pour le traitement vidéo est issue du projet VPM (Video Processing Module), une extension du projet Netpro. Les mécanismes d'interconnexion de la plate-forme de conversion de protocoles basés sur la norme AMBA ne supportent pas des communications complètement génériques. En conséquence, il faut adapter le matériel lorsqu'on change le nombre de modules du système. Donc, ce deuxième objectif impliquera la création d'une nouvelle méthode d'interconnexion générique pour relier des modules basés sur la norme AMBA et

pouvant opérer jusqu'à 500MHz dépendant du nombre de modules connectés sur le bus. La combinaison des deux objectifs permet la création d'une nouvelle méthode de conception utilisant le meilleur de chaque langage parmi un ensemble de langages disponibles. Dans ce cas, l'interconnexion des modules du système se fera sans exiger de changement sur ces mêmes modules. Le raffinement des communications des modules dans la méthode de conception sera déjà réalisé par un bus de haute performance. Il ne restera qu'à faire un raffinement au niveau des modules. Le canal de communication suivra la norme AHB d'AMBA. La plate-forme développée servira de référence pour le développement d'autres applications utilisant un environnement de conception hétérogène.

1.2 Contributions

L'objectif de ce travail est la création d'une plate-forme générique de haute performance avec une méthode à double profilage permettant de réaliser des systèmes pour différentes classes d'applications.

Les contributions majeures sont les suivantes :

- La création d'une méthode de conception hétérogène à double profilage. Cette méthode inclut une étude sur les langages de conception en fonction de leurs capacités et des niveaux d'abstraction. Dans le cadre de cette méthode, nous avons conçu différents adaptateurs permettant la connexion entre les langages.
- La création de trois méthodes d'interconnexions basée sur le protocole AMBA soit la méthode de connexion circulaire, par mémoire et par encodage de priorité. La dernière méthode permet une interconnexion multi fréquentielle des différents modules avec une mémoire principale opérant autour de 500MHz. Elle constitue un bus avec un adaptateur générique auto synchronisant qui ne requiert aucune programmation.
- La conception et l'analyse de l'architecture du convertisseur de protocole. À partir de analyse, il y a été possible d'élaborer une nouvelle structure d'interconnexion.

- À partir de la première plate-forme, nous avons conçu une nouvelle plate-forme pour le traitement et la transmission vidéo utilisant la méthode d'interconnexion des modules par encodage de priorité.

1.3 Organisation du mémoire

La première partie du chapitre 2 sera une introduction aux systèmes embarqués et présentera les langages de modélisation avec un cheminement typique de conception. À partir de ces informations, nous élaborerons une méthode de conception utilisant le meilleur de chaque langage. Par la suite, nous ferons une exploration des différentes plates-formes existantes dédiées au traitement vidéo. Aussi, nous discuterons de la plate-forme de conversion de protocoles avec son fonctionnement. De plus, les différents standards (protocoles) de communication feront l'objet d'une description sommaire. Nous discuterons notamment les normes AMBA, OpenCore et les mécanismes de connexion de la plate-forme Nexperia.

Le chapitre 3 présente le concept de plate-forme SoC et les éléments essentiels pour la connexion des langages entre eux. D'après l'analyse, nous pourrons connaître leurs caractéristiques communes et nous présenterons trois méthodes d'interconnexion entre les langages. Par la suite, nous élaborons des adaptateurs spécifiques aux langages permettant une simulation hétérogène. À partir des informations recueillies au chapitre 2 et 3, nous créerons une méthode de conception à double profilage à travers tous les langages présentés, c'est-à-dire le profilage d'un code sur deux niveaux d'abstraction différents mais complémentaires. De même, les méthodes d'analyse des plates-formes seront discutées. La méthode sera expliquée par l'algorithme de Wiener et découlera des résultats de simulations, de profilages et de comparaisons entre les langages.

Le chapitre 4 traitera de la réalisation d'un bus de haute performance. Nous présenterons le modèle d'interconnexion avec le bus AHB. Nous expliquerons les problèmes de synchronisation des horloges avec leurs phases. Suite à cette analyse, nous approfondirons trois méthodes d'interconnexion soit les connexions circulaires, par

mémoire et par encodage de priorité. De même, la création d'un adaptateur générique AHB sera présentée. Nous éliminerons les goulots d'étranglement de la plate-forme sur des flots de traitements avec le bus AHB. La fréquence d'opération supportée pourra atteindre 500MHz selon le nombre de modules lui accédant. L'adaptateur a été conçu à l'aide des bibliothèques de modules d'une technologie CMOS 0.18 micron. Enfin, le chapitre 5 présentera nos conclusions. Nous résumerons l'ensemble des travaux réalisés et présenterons les travaux futurs dans le cadre de cette recherche.

CHAPITRE 2
L'ÉTAT DE L'ART : OUTILS DE CONCEPTION, PLATE-FORME SoC POUR LE VIDEO ET INTERCONNEXIONS SoC

2.1 Introduction à la conception des systèmes embarqués

Les systèmes embarqués sont composés de différents modules matériels et logiciels. Il faut avoir les éléments adéquats permettant de réaliser une application souhaitée. La figure suivante présente la problématique de conception [16].

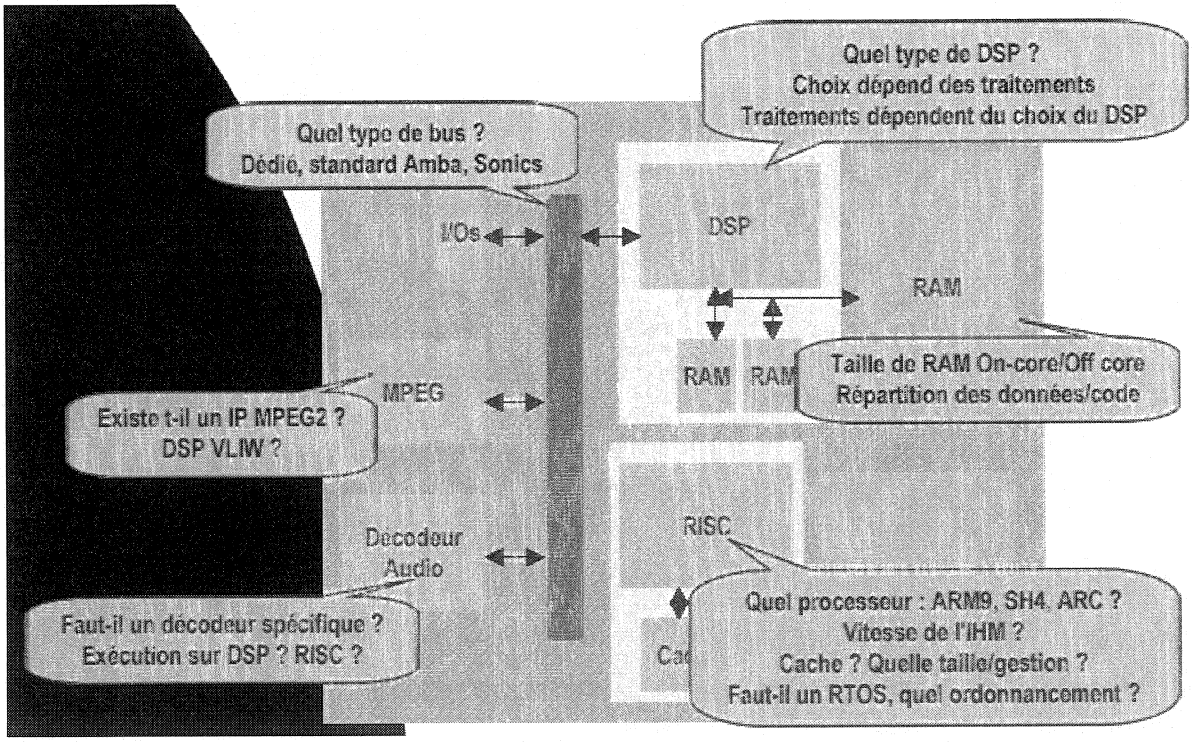


Figure 2-1 Illustration du Co-design

Une exploration rapide de plusieurs architectures (RISC, FPGA, ASIC, IP, DSP, ASSP, Analogique …) est requise et peut être possible grâce aux outils de codesign. L'évaluation des outils de conception et des méthodes qui y sont associées sera nécessaire pour connaître leurs capacités dans la réalisation d'un système sur puce. Il y a beaucoup d'outils sur le marché. Ils sont offerts à divers prix et tous ne sont pas bons à

tous les niveaux d'abstraction [29]. De plus, la diminution du temps de conception peut être réalisée en combinant judicieusement les langages entre eux. Idéalement, il faut des outils nécessitant un court temps de simulation de la spécification à l'implémentation physique. La figure suivante compare le temps de simulation pour différents niveaux d'abstraction. On remarque que le temps de simulation est plus court en logiciel qu'en matériel (RTL). Il faut simuler le matériel et le logiciel en même temps. Nous utilisons le VHDL pour une simulation du matériel et un modèle de jeux d'instruction (ISM pour Instruction Set Model) pour une simulation du logiciel dans un système embarqué. Dans un modèle ISM, le processeur est décrit en terme des instructions qu'il supporte.

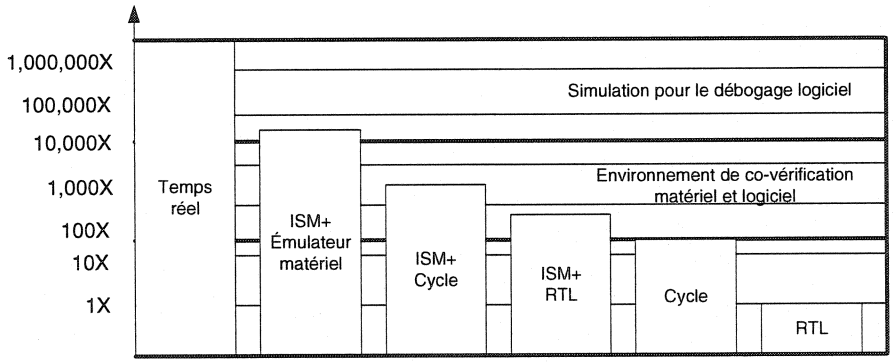


Figure 2-2 Comparaison des niveaux de performance [23]

Le développement des algorithmes, ainsi que leurs spécifications et leur implémentation nécessitent plusieurs outils. Dans ce qui suit, nous mettrons l'emphase sur les outils suivants: Matlab, SystemC, Modelsim, Seamless. Mais avant tout, il est important de connaître un flot de conception typique tel qu'illustré à la figure suivante tirée de [7]. La plupart des outils sont basés sur le C/C++ et ils peuvent supporter des étapes différentes du processus de la conception. Ici, il est important de souligner l'importance des méthodes fondées sur le langage C/C++ tel que SystemC dans la conception d'un SoC. Pour cette analyse, il y a quatre phases [37]: l'architecture, l'implémentation, la vérification et autres éléments. Tout d'abord, l'architecture permet de définir un système selon les spécifications du client et tous les éléments nécessaires avant la conception. Il y a aussi le choix des interconnexions et des modules IP.

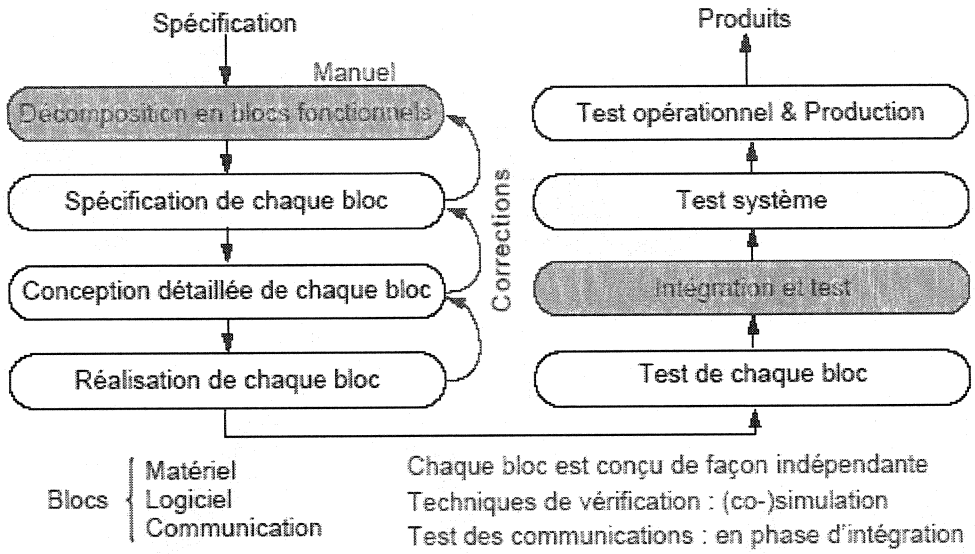


Figure 2-3 Flot typique de conception

Ces derniers peuvent être un CPU, un DSP, un noyau Ethernet etc. Entre l'architecture et l'implémentation il y a la spécification exécutable. Par exemple, SystemC permet la création d'un prototype en C/C++ exécutable, qui fournit une représentation flexible exprimée au niveau système. De plus, il peut y avoir des modules matériels. Aussi SystemC permet la séparation entre l'infrastructure des communications et les modules de traitement. Cette séparation permet une plus grande flexibilité. Ensuite, il y a l'implémentation englobant tous les niveaux d'activité des blocs et des supers blocs incluant le niveau bloc du design et le test de l'intégration du système. Dans cette phase, la plupart des bancs d'essais n'ont pas avantage à utiliser du C/C++ parce qu'il n'y a pas un grand contenu algorithmique.

La seconde phase est la vérification. Lors de la conception de SoC de grande complexité, la productivité du processus de vérification est un grand obstacle pour rencontrer la demande de mise en marché. Ici, tout comme pour l'architecture, le C/C++ est utilisé pour augmenter la productivité et les aspects de performance [37]. Les nouveaux outils séparent les fonctions de communication permettant de modéliser un système facilement et offrent un gain de productivité global. Le dernier aspect est constitué des autres

éléments qui peuvent survenir après une vérification fonctionnelle [37]. D'autres facteurs importants dans le choix d'un outil sont sa capacité et son coût récurrent ou non. La figure suivante présente sous une forme normalisée les prix des outils. Cette comparaison normalisée est basée sur des estimations des prix du marché en 2003 receuillis de diverses source sur le réseau et dans la litérature techico-commerciale.

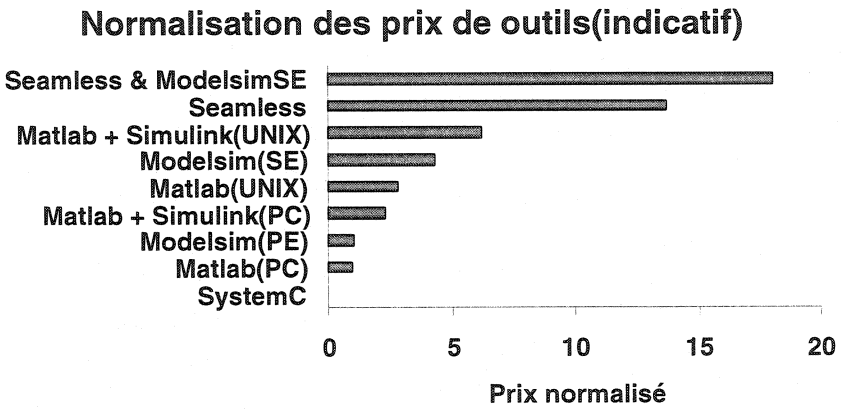


Figure 2-4 Prix normalisés des outils

Nous constatons que les prix des outils sur PC sont deux fois et demie moins cher que sous Unix. Tous les outils mentionnés sont disponibles sous UNIX ou PC, sauf Seamless qui n'est pas disponible sur les systèmes d'exploitation windows. Il est important de mentionner que les licences sont attribuées par utilisateur ou par machine et qu'elles font l'objet de frais récurrents. Lors d'une simulation, il est généralement important d'exploiter au maximum les ressources matérielles et le minimum des licences logicielles (jetons). L'utilisation maximale d'une licence d'une machine peut être réalisée avec des connexions TCP/IP permettant une simulation à distance partagée où la licence est exploitée par plusieurs utilisateurs.

Nous avons donc intérêt à utiliser SystemC dans notre méthode, car il est gratuit. La section suivante présente principalement les outils HDL Designer, Matlab, SystemC et Modelsim.

2.2 Aperçus des outils

2.2.1 HDL Designer

La conception des SoC requiert de la programmation au niveau logiciel et/ou matériel. Il est important d'avoir un outil permettant l'intégration des modules sous différentes formes de textes ou de graphiques et permettant l'incorporation de divers langages comme VHDL, C et C++. Un mode graphique permet de visualiser plus facilement le fonctionnement d'un module en représentant la fonctionnalité voulue sous forme graphique, par exemple, une machine à états finis. L'outil HDL Designer [31] servira de contrôleur de version, de gestionnaire de bibliothèques et il fournit un ensemble d'autres fonctionnalités connexes à la gestion d'un projet de conception d'un système embarqué. De plus, son interface permet de démarrer une simulation avec d'autres langages. Une représentation graphique et modulaire d'un système est plus facilement compréhensible lors de la conception. Nous l'utiliserons dans le cadre de notre projet.

2.2.2 Matlab

La modélisation d'un système peut être faite avec Matlab [26]. Cet outil permet la modélisation à plusieurs niveaux d'abstraction. Il a son propre flot de conception. Il permet un prototypage rapide avec les FPGA. Il a six champs d'applications:

- Calcul Technique
  - Analyse, visualisation et développement d'algorithmes
- La conception du contrôle
  - L'outil fournit des modèles de design de base pour le contrôle;
  - Il permet de décrire l'algorithme par la création d'un modèle et il permet la simulation de ce modèle;
  - Il permet ensuite le prototypage rapide;
  - Il permet enfin la génération du code pour les systèmes embarqués.
- Traitement de signal (DSP) et conception des système de communication
  - Il permet d'effectuer la conception au niveau système pour le DSP et les systèmes de communication;

- Il permet de décrire chaque algorithme par la création d'un modèle et il permet la simulation de ce modèle;
- Il permet ensuite le prototypage rapide;
- Il permet enfin la génération du code pour les systèmes embarqués.

➢ Tests et mesures
- L'outil supporte la collecte des données à partir de plate-forme matérielle.

➢ Traitement d'images
- L'outil supporte l'acquisition et l'importation d'images et leur analyse;
- Il permet de perfectionner et de développer des applications.

➢ Analyse et modèle financier
- Il supporte l'analyse, la modélisation, la simulation et l'optimisation des données financières.

Matlab permet un raffinement progressif partant d'un haut niveau d'abstraction en allant vers le bas. Il permet deux types de génération de code automatique. Premièrement, l'outil Real-Time Workshop peut générer automatiquement du code C ANSI à partir d'un modèle Simulink pour le téléchargement vers un processeur DSP ou un processeur embarqué. De plus, des mises en œuvre matérielles peuvent être générées par le générateur de système de Xilinx. Matlab présente des avantages pour le développement automatisé ciblé FPGA et des applications DSP. La figure suivante présente sa méthode de conception.

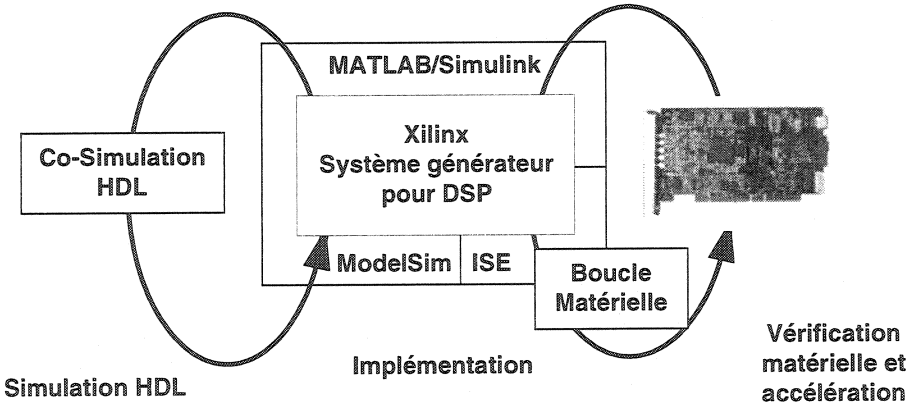


Figure 2-5 Flot de conception de Matlab pour DSP

Cette méthode permet le développement sur FPGA. D'autre part, la traduction du code Matlab en C/C++ peut être réalisée par l'outil MCC et le code produit peut être récupérédans une simulation hétérogène. De plus, Matlab fournit des fonctions API permettant au C de communiquer et de démarrer Matlab.

2.2.3 SystemC

Un autre langage important lors de la conception est SystemC. C'est une bibliothèque en C++ orienté objet pour la modélisation logicielle et matérielle. Cette section discute des possibilités d'utilisation de ce langage dans la conception d'un SoC. Il possède des modules, des ports et des canaux de communication. L'intérêt de ce langage est de pouvoir raffiner les modules et les méthodes de communication individuellement. La figure suivante présente l'interconnexion possible entre deux modules.

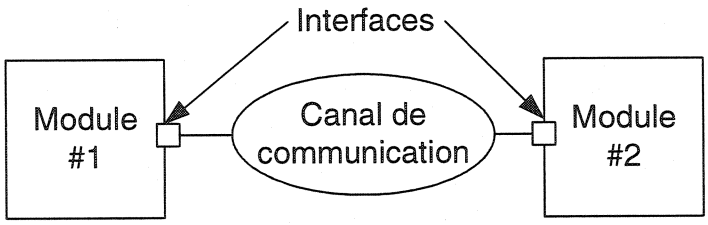


Figure 2-6 Illustration des interfaces

Il possède des processus, des ports, des types de données et des signaux d'horloges. Les processus offert en SystemC sont sc_method, sc_thread, sc_cthread. Le premier permet d'exprimer des processus combinatoires. Le deuxième et le troisième permettent la description d'une exécution séquentielle. Le dernier a les mêmes caractéristiques que le deuxième processus, mais il a uniquement une horloge comme liste de sensibilité. Nous trouvons dans ce langage les notions de traitement combinatoire et séquentiel comme dans le langage VHDL. Il permet de spécifier les ports d'entrées, les ports de sorties et les ports bidirectionnels. Ces derniers permettent une communication entre les modules. Le type de données matérielles est disponible grâce à l'objet sc_signal<T>. Des signaux pour la manipulation de bits, le traitement de vecteurs et des entiers sont aussi possibles. Un autre signal important dans la conception d'un système est l'horloge. Ce dernier exprime l'information temporelle et permet la synchronisation des processus entre eux.

SystemC permet six niveaux d'abstraction : UTF (UnTimed Functional), TF (Timed Functional), ISS (Instruction Set Simulator), CA (Cycle Accurate), BCA (Bus Cycle Accurate), PA (PIN accurate). La figure suivante présente un flot possible pour le raffinement d'une spécification [13].

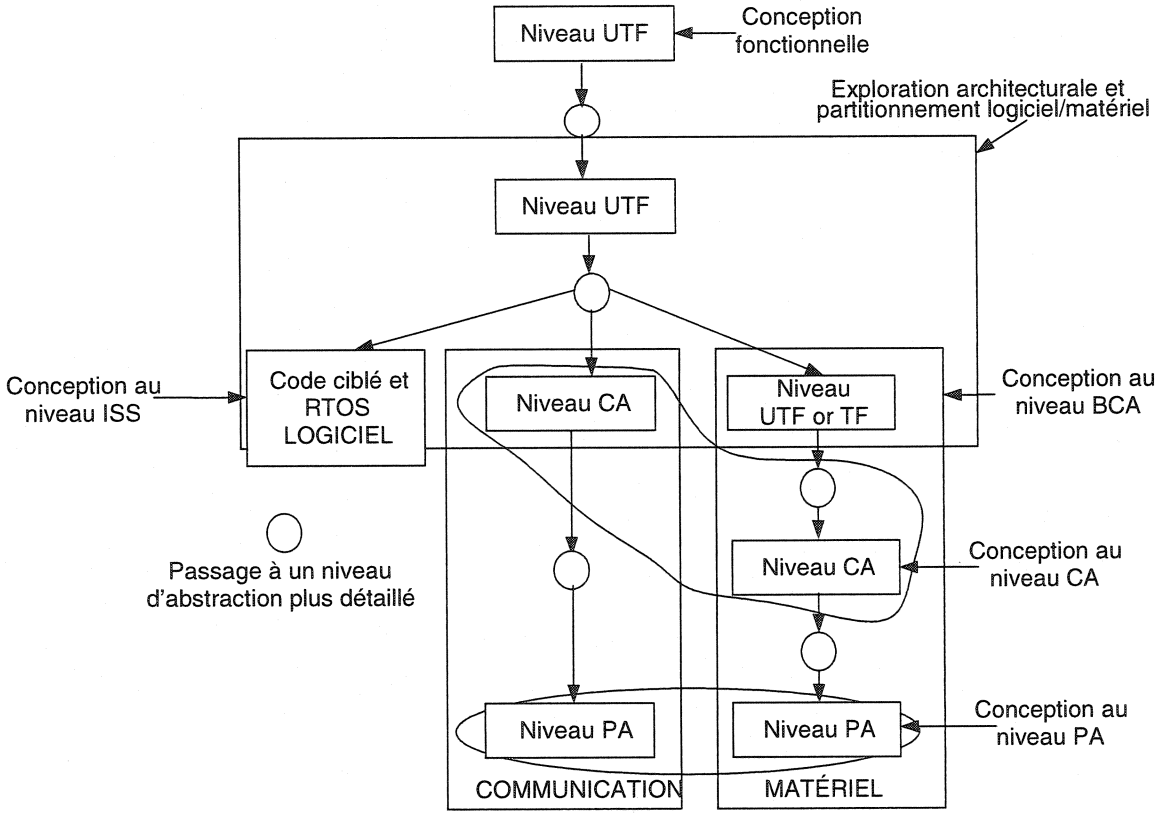


Figure 2-7 Flot de conception avec SystemC

Le plus haut niveau UTF permet une conception fonctionnelle d'un système. De plus, une exploration rapide de plusieurs architectures, impliquant le partitionnement logiciel et matériel, peut être réalisée à partir des niveaux d'abstraction UTF, TF et ISS. Le niveau ISS permet le raffinement de la partition logicielle à partir du niveau TF. Le raffinement matériel se fait à trois niveaux. Tout d'abord, il faut faire une synthèse des communications à l'aide du niveau BCA. Ce dernier permet de décrire un médium de communication au cycle près tout en conservant la simplicité du niveau UTF. Ensuite, en synthétisant celui-ci, nous obtenons une description au niveau CA. À ce niveau, les

communications ne sont pas encore détaillées entre les modules et les communications. Celui-ci permet de diminuer le temps de simulation tout en ayant une simulation précise au cycle près. En terminant, lorsqu'on a raffiné les communications abstraites, il faut passer au niveau PA qui exprime tous les signaux au niveau broche. Un autre aspect important lors de la conception est d'obtenir des informations permettant l'analyse de notre système. Le prochain outil permet cette analyse.

2.2.4 Seamless

Une simulation hétérogène de plusieurs outils peut être effectuée par Seamless [30]. Il offre la conception de système à plusieurs niveaux d'abstraction. Une simulation mixte VHDL et C/C++ est possible par l'API C-Bridge comme l'illustre la figure suivante [22].

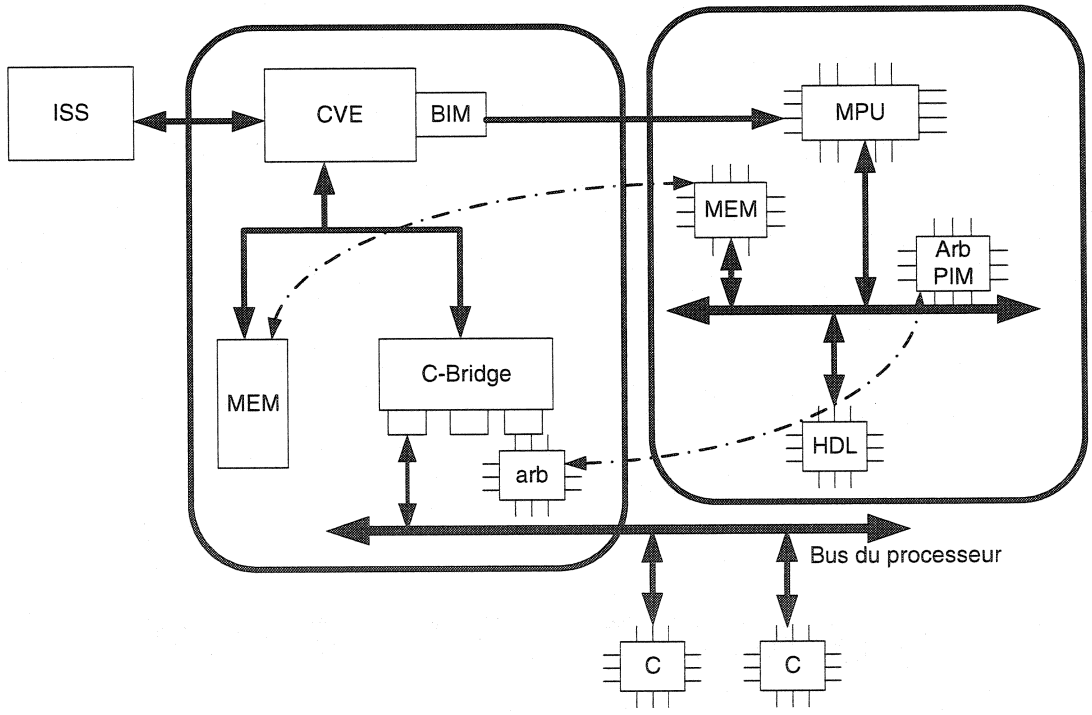


Figure 2-8 Simulation avec C-Bridge

Par exemple, une mémoire (MEM) peut être modélisée en C et ensuite transférée dans le modèle RTL. Seamless permet deux niveaux d'abstraction soit le BIM (Bus Interface Model) et le PIM (Pin Interface Model). Le BIM permet d'être à un plus haut niveau d'abstraction que le PIM. Il permet de réaliser des simulations sans notion de temps. Les

méthodes de communication sont des requêtes et des accusés de réception permettant d'avoir un bus transactionnel. Le deuxième type (PIM) permet la connexion d'une interface VHDL au C/C++. Lors du raffinement, il est important de pouvoir garder le plus longtemps possible un niveau d'abstraction de haut niveau pour avoir un temps de simulation plus rapide. Nous pouvons transformer notre modèle de C/C++ en VHDL tout en conservant le reste de l'architecture. Une autre approche de conception est aussi possible avec la dernière version de Seamless (version 5) [11]. En effet, il est possible de simuler sans simulateur VHDL, c'est-à-dire uniquement à partir d'un simulateur ISS et de SystemC. La figure suivante illustre ce propos.

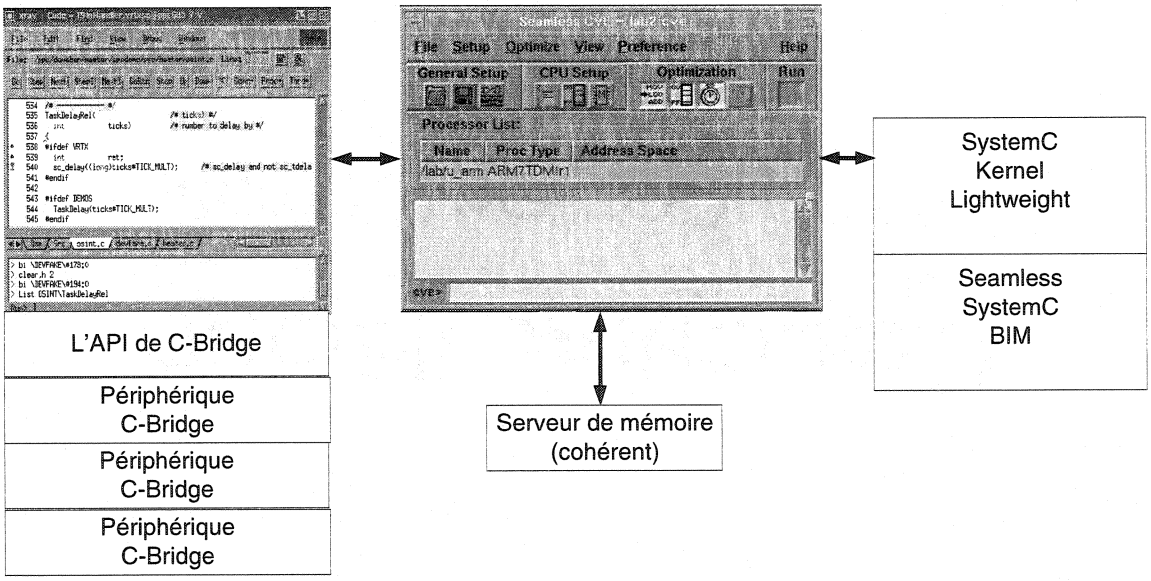


Figure 2-9 Seamless avec SystemC

L'idée est de réaliser les blocs matériels en C/C++ plutôt qu'en VHDL. Il y a plusieurs avantages à cette méthode. Tout d'abord le temps de simulation est réduit. La création d'une modélisation est plus simple en C qu'en VHDL. Il y a aussi une possibilité de passer d'un haut niveau d'abstraction en C/C++ à un modèle implanté en RTL. L'API C-Bridge permet la simulation d'un module créé en C/C++ représentant un comportement matériel. Par exemple, le matériel modélisé en C/C++ peut correspondre à un niveau d'abstraction TF. La compagnie Mentor Graphics fabricant Seamless intègre Modelsim, SystemC, et les modèles ISS ensembles. De plus, cette méthode permet d'avoir une

spécification exécutable. Tous les éléments du système sont écrits en C/C++. Lors d'un raffinement, le code en C/C++ pour le logiciel peut être mis dans un processeur. Le matériel peut être transformé de SystemC à VHDL. Un autre avantage de la nouvelle version est la possibilité de faire de l'analyse de performance (essentiel au codesign). En effet, il y a quatre nouvelles analyses disponibles [30]. Premièrement, un profileur de code permet d'examiner le temps d'exécution d'un logiciel. Deuxièmement, il est possible de caractériser l'utilisation de la bande passante d'un bus. Troisièmement, il est possible de caractériser le délai d'arbitration d'un bus, c'est-à-dire combien de temps un maître doit attendre pour avoir le bus. Quatrièmement, il est possible d'évaluer l'information concernant les données sur les transactions en mémoire, soient les accès manqués en cache ou les accès réussis.

2.2.5 Modelsim [32]

Le prochain outil nommé Modelsim permet une simulation à un plus bas niveau d'abstraction. ModelSim est un simulateur VHDL largement répandu et un simulateur mixte combinant le VHDL et le Verilog. Les produits ModelSim® ont une architecture unique basée sur des technologies comme *Optimized Direct Compile* pour des compilations et des simulations rapides par le *Single Kernel Simulation* (SKS). De plus, il incorpore le langage TCL/TK pour une grande flexibilité et une vérification rapide. Une personnalisation facile est permise par le TCL/TK. Ce dernier permet de développer des interfaces graphiques. Il possède également des connexions FLI (Foreign Language Interface) / PLI (Programming Language Interface) permettant une intégration du C/C++. Les nouvelles versions possèdent un débogueur intégré pour faciliter la vérification d'un système. Modelsim inclut un comparateur de chronogramme, des analyseurs de performance et la capacité de donner la couverture de code. Le comparateur de chronogramme permet d'augmenter la vitesse de vérification entre deux simulations pour identifier les erreurs. Il permet aussi d'analyser la vitesse de simulation en identifiant les congestions. Celui-ci peut découvrir les codes qui ne sont pas efficaces, les cellules d'une bibliothèque non accélérées, des signaux inutiles dans la liste de sensibilité, etc. La couverture de code permet de connaître combien de fois l'exécution

d'un segment de code a été effectué. Le Signal *Spy* permet de voir à travers la hiérarchie. Il est possible de mettre un moniteur sur un signal et de le voir directement dans le banc d'essai.

2.2.6 SystemVerilog

Un autre outil nommé SystemVerilog [40][15] est une extension majeure de la norme IEEE 1364-2001 qui spécifie le langage Verilog. Il a été développé par Accellera pour améliorer la productivité dans la conception des circuits intégrés à grande échelle basés sur des IP et des systèmes de bus. Il supporte la modélisation à haut niveau et la vérification avec des assertions. Il permet un appel direct de fonctions en C/C++/SystemC en utilisant son *Direct Programming Interface* (DPI). SystemVerilog possède les types de données comme le C et la possibilité de créer des classes en ayant aussi les signaux au niveau matériel. De plus, il est possible de créer ses propres structures. Des notions d'interfaces existent, ainsi que des primitives de niveau système et des mécanismes de communication tel que les boîtes aux lettres et les sémaphores, etc.

2.2.7 UML

Un autre outil permettant la modélisation à un niveau d'abstraction plus élevé est le *Unified Modeling Language* (UML) qui a été créé par OMG (Object Management Group) pour le développement des systèmes logiciels. La compagnie Fujitsu a développé une nouvelle méthode de conception utilisant UML et le langage C pour la création d'un SoC [17]. UML est un outil de modélisation visuel avec un haut niveau d'abstraction permettant une spécification exécutable d'un système. Pour plus de détails sur son fonctionnement, [28] permet de connaître davantage ses capacités et ses lacunes pour les spécifications d'un système embarqué ainsi que sa conception.

2.3 Convertisseur de protocoles

La communication entre les systèmes embarqués d'aujourd'hui est basée sur des protocoles de communication distincts. Chacun d'eux permettent la communication entre différents périphériques matériels tels que les ordinateurs, les caméras, les systèmes de

télésurveillance, etc. Notre objectif dans ce cas-ci est une communication entre le protocole IEEE 802.11 (Ethernet) à IEEE 1394 (Firewire), et vice versa. Le convertisseur fait correspondance à une couche précise du modèle OSI les différents protocoles de communication. L'illustration suivante présente la plate-forme d'un convertisseur de protocoles.

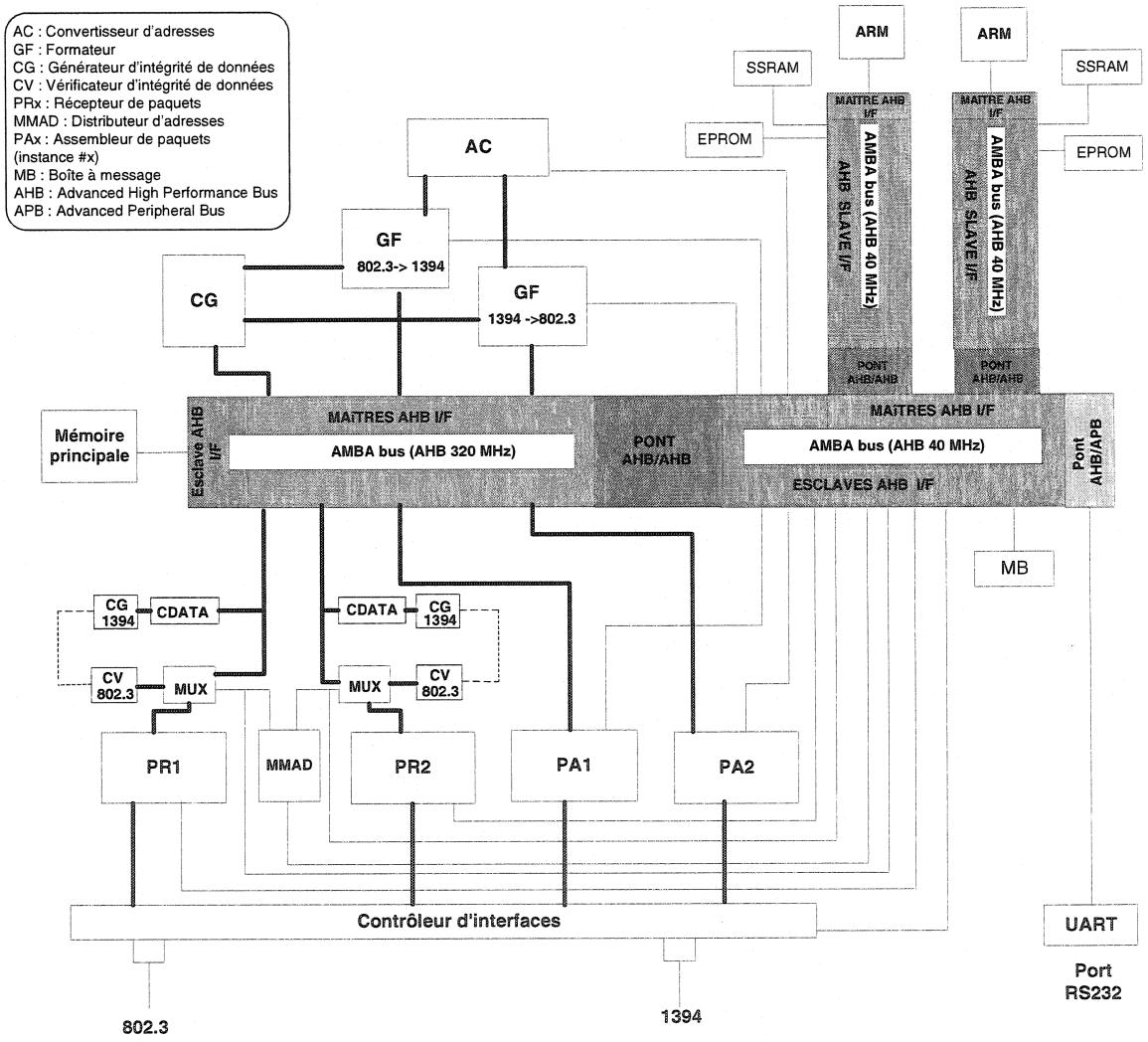


Figure 2-10 Plate-forme du convertisseur de protocole

Nous avons deux modules de réception de paquets, soit PR1 et PR2, ainsi que deux modules de transmission de paquets, PA1 et PA2. Il y a quatre processeurs, soit 2 ARMs avec 2 GFs et huit principaux modules matériels. Nous employons une mémoire centrale

avec un port d'accès. La configuration de cette plate-forme est possible par le UART ou par un lien Ethernet. Nous avons une plate-forme *multi-threaded*. C'est un système de conversion bidirectionnel. Le logiciel est exécuté sur un ARM7TDMI connecté par un bus de commande AMBA AHB. Selon les besoins de l'application, un deuxième ARM peut faire un traitement sur un paquet tel que la reconnaissance de protocoles. Les processeurs ARMs ont été isolés avec leur propre bus AMBA pour réduire au minimum le trafic sur le bus de contrôle. Les deux ARMs ne peuvent pas accéder en même temps au bus de commande.

Tout d'abord, un paquet arrive d'un lien physique par le PR. Ce dernier vérifie s'il peut identifier le protocole, sinon il envoi une requête au ARM pour un décodage logiciel. Le PR mettra le paquet en mémoire. Par la suite, le formateur (GF, pour *general formatter*) le prendra pour le convertir avec son entête à un autre protocole à partir de ses différentes couches. Les résultats seront mis dans la mémoire principale. L'assembleur de paquet rassemblera les entêtes et les données du paquet pour les transmettre dans un autre protocole. Le ARM de contrôle permettra de suivre l'état du paquet en cours de traitement. De plus, il contrôlera chaque module du système pour leur indiquer leur tâche à suivre. Le MMAD (Main Memory Address Distributor) permet l'attribution des plages mémoires occupées par les paquets. Il y a aussi une boite aux lettres permettant la communication entre les deux processeurs ARM. Le bus de donnée à haute vitesse permet le lien entre les modules et la mémoire de façon concurrente. Les paquets peuvent être acheminés de la mémoire aux modules et vice versa sans avoir un problème de contention. Chacun d'eux peut réaliser une opération sur le paquet et le remettre en mémoire. De plus, c'est une mémoire à un seul port qui permet la réduction de la surface comparativement à une mémoire multiport. Un autre module nécessaire est le CG (*Checksum Generator*) qui permet de savoir s'il y a eu des erreurs sur les paquets. L'interface contrôleur permet à la plate-forme et de s'interfacer aux protocoles traités. Dans la mise en œuvre envisagée, le bus de contrôle opère à 40MHz et le bus de données à 320MHz. Le GF et le CA opèrent à 80MHz, tandis que les autres modules opèrent à

40MHz. Les modules sur le bus de données sont des maîtres pouvant initialiser des requêtes. Les modules du bus de contrôle sont des esclaves recevant des ordres provenant des ARM. Les ARM peuvent accéder à la mémoire à travers deux pont AHB/AHB. Le pont AHB/APB permet l'interconnexion au système des périphériques lents. Plus particulièrement, le convertisseur traitera une transmission de paquets de type vidéo.

2.4 Autres architectures des plate-formes SoC vidéo

Un des objectifs de ce travail est de définir une plate-forme générique permettant une application de traitement et de transmission vidéo. À titre de revue de littérature, nous allons présenter deux plate-formes vidéo, soit la VC01 de Alphamosaic et la Nexperia-DVP de Philips.

2.4.1 VC01

Alphamosaic offre un circuit intégré VCO1 [2] permettant le traitement vidéo à faible consommation de puissance. Il est programmable et une bibliothèque logicielle est disponible pour réaliser des fonctions tel que l'évaluation du mouvement, DCT, l'analyse en ondelettes, l'application d'opérateurs de morphologie mathématique, etc. Les applications visées peuvent être le téléphone vidéo, la vidéo sans fils, les assistant digitaux etc. Il est basé sur l'utilisation de processeurs DSP avec unités VLIW vectorielles de traitement comprenant 16 chemins de données en parallèle (16 processeurs) opérant jusqu'à 125MHz. De plus, il est associé à un processeur scalaire RISC de 32 bits. Il a un registre 2D agissant comme une fenêtre sur l'image, permettant au maximum un groupe de pixels de 64x64. Il peut faire 6 milliards d'opérations scalaires équivalents (BOPS). Cette architecture possède une mémoire interne de 8 MBits. La figure suivante présente une vue de la structure interne de ce circuit intégré [3].

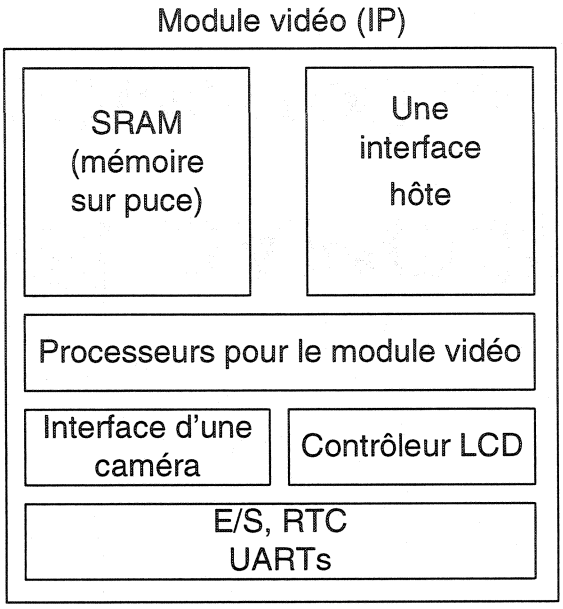


Figure 2-11 Schéma bloc de la puce d'Alphamosaic

Il supporte le système d'exploitation en temps réel tel Nucleus [1]. VC01 exploite la dernière technologie CMOS 0.13 micron.

2.4.2 Nexperia

Les plates-formes Nexperia de la compagnie Philips Semiconductors représentent une famille de solutions SoC flexibles permettant des applications multimédia. L'approche de Phillips fut de développer un système flexible à travers la programmation et l'utilisation de blocs IP ainsi que d'une architecture de base. Le PNX-8525 et la PNX-8550 sont des exemples de circuits intégrés qu'on retrouve sur la Nexperia-DVP. L'architecture de base de la DVP est représentée par la figure 2-12 [5].

Les principaux éléments sont les processeurs, les blocs IP, le réseau de connexion et l'interface à la mémoire principale. Il y a deux processeurs, le premier est un processeur MIPS servant pour le contrôle de la plate-forme et le deuxième est un processeur multimédia pour le traitement vidéo.

L'architecture comprend trois niveaux abstractions soit les règles logicielles et matérielles, le niveau de transaction des modules (DTL) et la connexion réseau.

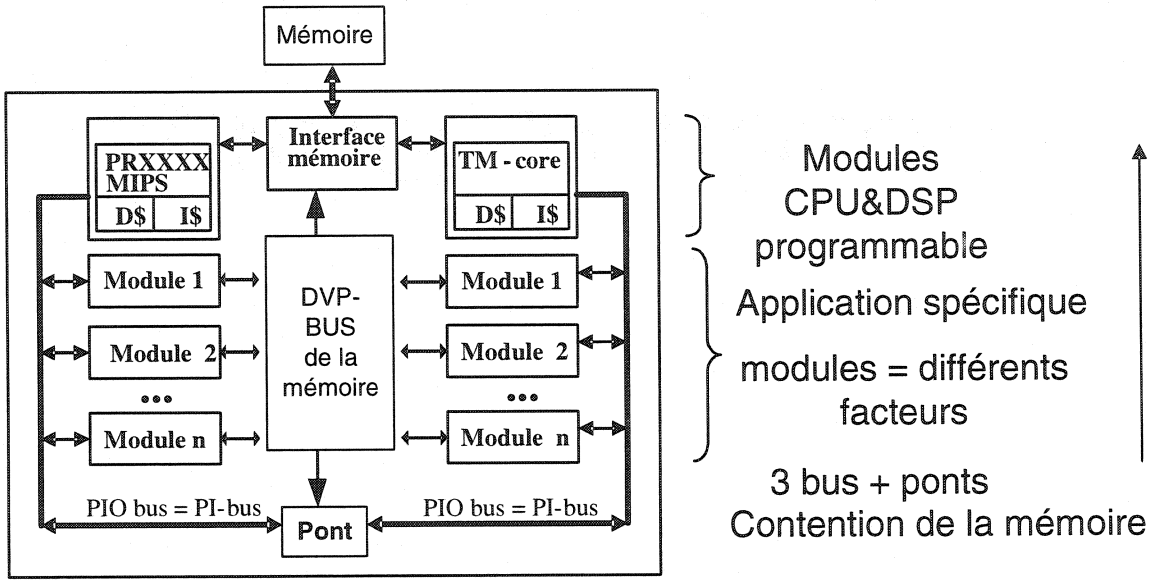


Figure 2-12 DVP-based IC

Le premier sert à définir le logiciel vu comme du matériel. Le deuxième permet le transfert entre les modules et le dernier est le niveau bus de communication. La prochaine section discute plus en détails des interconnexions entre les modules d'un système sur puce. Tout comme le VC01, la plate-forme possède un système exploitation. Il y a toujours un processeur de contrôle et un autre pour le traitement des données. Dans la figure 2.12, on observe qu'il y a un bus dédié à chaque type de processeur et ils sont reliés par un pont.

2.5 Interconnexion d'une plate-forme SoC

L'interconnexion des modules d'une plate-forme SoC requiert une capacité d'échanger des données qui satisfait la bande passante requise par l'application visée. Nous traiterons trois manières permettant la réalisation des interconnexions d'une plate-forme SoC, dont deux se basent sur des normes de communication existantes: celles de la plate-forme Nexperia , du Star-IP bus utilisant AMBA et finalement OCP d'IBM.

2.5.1 Interconnexion d'une plate-forme Nexperia-DVP

La connexion d'un module avec la mémoire se fait en utilisant deux types de port de communication. Nous avons le MTL (*Memory Transaction Level*) et le DTL (*Device Transfer Level*). La figure 2.13 permet l'illustration des ports MTL dans un SoC de Nexperia.

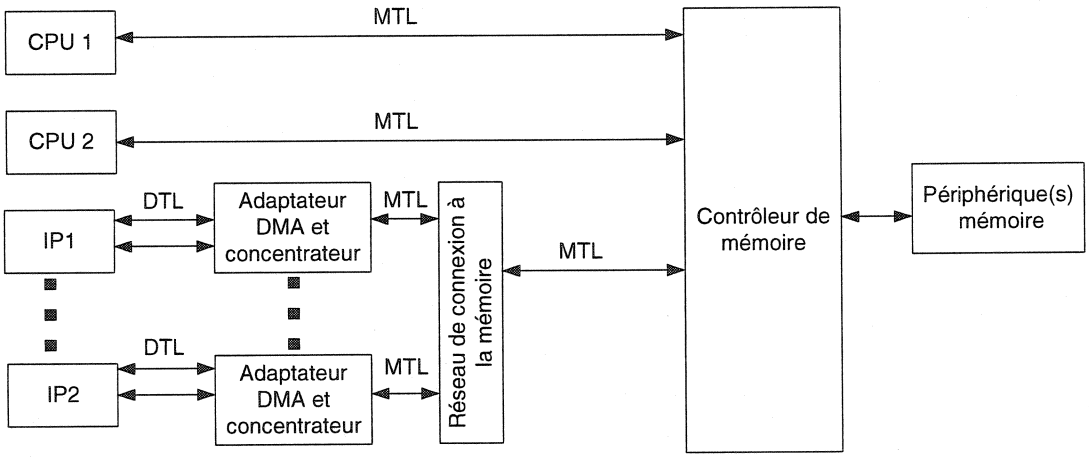


Figure 2-13 Application des ports MTL dans un SoC de Nexperia

Le MTL est un protocole de communication optimisé pour un contrôleur de mémoire DRAM. La connexion à la mémoire utilise un PMAN (*Pipeline Memory Access Network*). Le protocole DTL précède et ressemble à l'interface VCI de VSIA [43]. La différence est qu'il y a une abstraction de certains aspects du système comme la bande passante de la mémoire, la longueur optimale des transactions, etc. Plus précisément, il n'y a pas de détails relatifs au protocole de communication dans le développement des IP.

Un autre aspect important est la connexion réseau au niveau du bus. La figure 2.14 présente le système d'états et le contrôle des périphériques de la Nexperia [5]. Il comprend un contrôleur d'état de réseau pour les modules et un réseau d'interconnexion pour la mémoire. Le réseau DCS permet l'implémentation d'îlots synchrones. Il est nécessaire d'avoir un adaptateur DTL-DCS. Chaque module peut opérer à une fréquence de 70 à 150MHz avec une horloge de DCS de 200MHz.

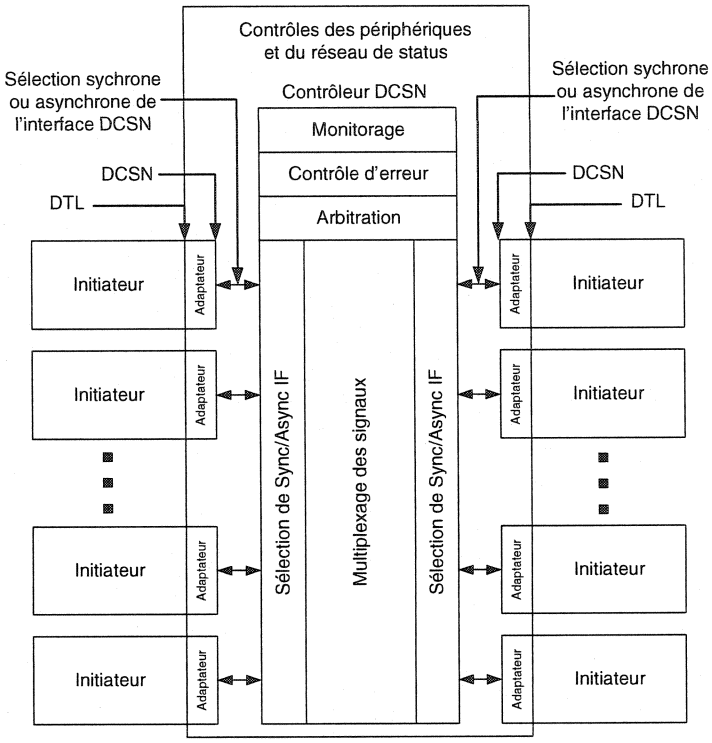


Figure 2-14 Illustration générale de l'interface pour les contrôles des périphériques

Il est configuré à chaque initiateur ou cible avec des mécanismes synchrone ou asynchrone. La bande passante du système peut être fixe ou dynamique selon l'application souhaitée.

2.5.2 Star-IP Bus

Une autre approche de conception des interconnexions est le noyau Star-Ip [4]. C'est une architecture de plate-forme basée sur des sous systèmes. Nous allons voir son application sur la plate-forme ARM(R) PrimeXsys^{TM} [9]. L'architecture est composée de sous systèmes embarqués constituant un module ARM, un protocole de communication AMBA et un système d'exploitation. AMBA est utilisé dans plusieurs champs d'application. La figure suivante présente ses éléments.

Nous avons plusieurs bus AHB avec plusieurs ponts et un processeur ARM. Nous pouvons connecter des périphériques rapides ou lents sur les bus AHB et APB

respectivement. AHB est un protocole à haute vitesse, tandis que le APB fonctionne à plus faible fréquence et du même coup à une plus faible consommation de puissance.

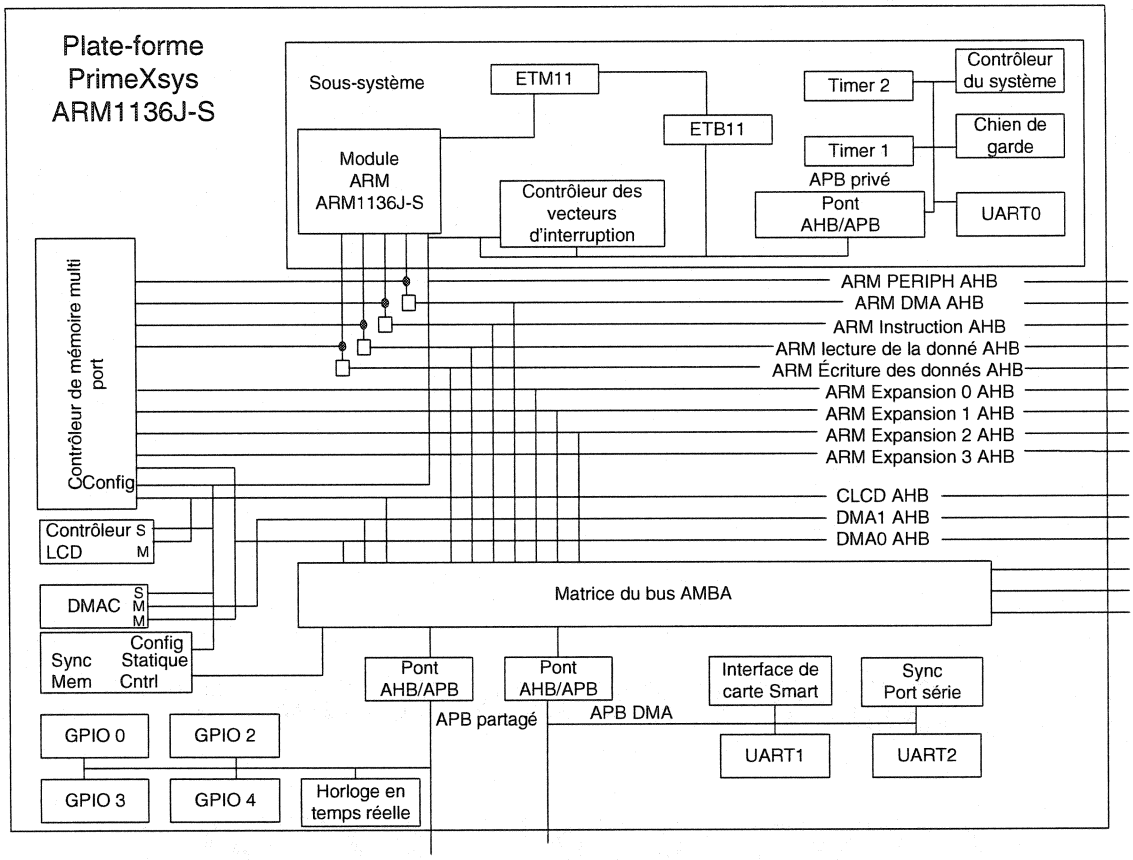


Figure 2-15 Plate-forme PrimeXsys

La norme AHB permet la connexion de 16 maîtres au maximum et d'autant d'esclaves que l'on désire. Le logiciel est exécuté sur le processeur. Dans cette architecture d'interconnexion, les modules compatibles AHB peuvent se connecter sur le bus. Il fonctionne avec un arbitre, des maîtres et des esclaves. Les maîtres initient les requêtes et les esclaves leur répondent. Il y a deux phases. La première est la phase d'adresse permettant de connaître ce que le maître veut et la deuxième est la phase de donnée pour écrire ou lire une valeur au module esclave. Dans cette plate-forme, une matrice de bus AMBA permet la communication simultanée de plusieurs maîtres sur le bus à des différents esclaves. La réduction du temps de conception est possible grâce à l'intégration de différentes plates-formes sur une puce utilisant AMBA.

2.5.3 Le bus CoreConnect

La réalisation des interconnexions entre les modules peut se faire par une méthode différente, soit celle développée par la compagnie IBM avec son bus CoreConnect [20]. Il a été initialement développé pour l'utilisation d'un processeur embarqué PowerPC 405 avec une technologie 0.5µm. Il possède trois niveaux hiérarchiques soit le *Processor Local Bus* (PLB), le *On-chip Peripheral Bus* (OPB) et le *Device Control Register* (DCR). Le premier sert de communication pour les processeurs avec une bande passante élevée. Le deuxième permet la connexion de périphériques opérant à basse vitesse. Le dernier sert à la configuration nécessaire pour certaines instructions du PowerPC. Son système d'interconnexion est illustré par le diagramme suivant.

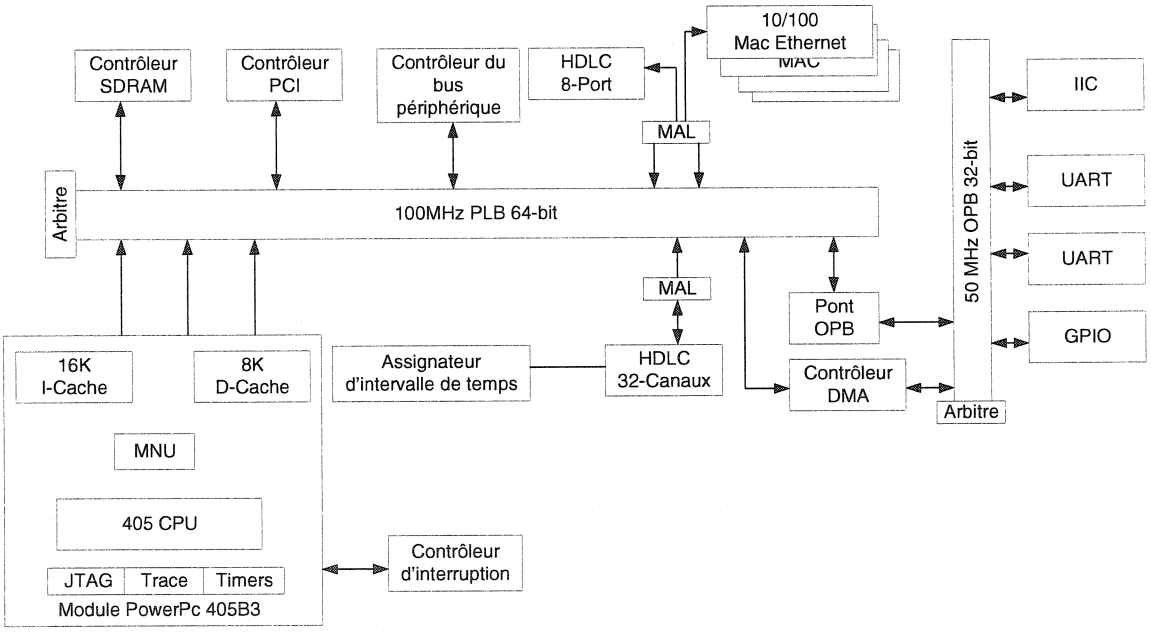


Figure 2-16 Diagramme bloc du PowerNp [21]

Il vise à créer un environnement de conception facilitant l'intégration de modules provenant de différents concepteurs. Tout comme les autres plates-formes, il utilise une plate-forme de référence pour le développement d'un nouveau système.

2.6 Conclusion

La réduction du temps de conception des systèmes complexes est réalisée par une méthode d'intégration des modules dans une plate-forme de référence. Il y a différentes manières de connecter les modules entre eux. Dans la revue de littérature, il y a presque toujours deux bus dans un SoC. Un bus permettant une connexion à haute vitesse pour les processeurs et un autre permettant la communication avec des périphériques opérant à basse vitesse. L'utilisation d'un pont permet la connexion des deux bus. Dans les plates-formes multimédia, nous retrouvons un bus de contrôle et un bus de traitement. Le premier permet la gestion de la plate-forme et le deuxième sert pour le traitement d'une application souhaitée. Pour atteindre une mise en marché rapide, les concepteurs utilisent des normes de communications entre les modules. Nous avons discuté particulièrement de la norme AMBA et CoreConnect.

De plus, la conception d'un système nécessite plusieurs langages ayant chacun leur caractéristiques. Ces derniers ne sont pas efficaces à tous les niveaux d'abstraction. Il est nécessaire d'avoir une méthode de conception exploitant le maximum de capacités de chaque langage. Ainsi, l'utilisation de plus d'un langage avec une méthode d'interconnexion de modules permet de réduire le temps de conception. La simulation des composants créée par des tiers sous différents langages avec un système d'interconnexion, par exemple AMBA, est possible grâce à un environnement hétérogène.

CHAPITRE 3
MODÉLISATION HÉTÉROGÈNE

Un objectif de ce travail de recherche est de créer un environnement hétérogène exploitant plusieurs langages de conception en utilisant le meilleur de chacun. La création de cet environnement permettra une utilisation de bibliothèques matérielles et logicielles dans un flot de conception non homogène. Cette méthode permettra la création d'une plate-forme de conception virtuelle de transmission et de traitement vidéo. Ce chapitre discutera de l'interconnexion des langages de modélisation permettant une simulation hétérogène. À partir des possibilités d'une simulation des composantes hétérogènes, nous développerons une méthode de raffinement progressif à double profilage appliquée à l'algorithme de Wiener [24]. Ce dernier est un filtre adaptatif 2-D pour la réduction du bruit sur une image. Le filtre Wiener permet d'avoir une image filtrée où le bruit présent dans l'image est atténué. L'atténuation du bruit est obtenue par l'équation suivante qui produit les pixels dans l'image de sortie en ajoutant à la moyenne locale une partie de la différence entre les pixels d'entrée et cette même moyenne locale calculée sur un voisinage d'une taille spécifiée par l'utilisateur.

$$F(k,l)=\mu+\frac{\sigma^2-v^2}{\sigma^2}(G(k,l)-\mu)$$

3.1 Plate-forme SoC

3.1.1 Concept

Le concept de plate-forme SoC n'est pas entièrement nouveau. Il a été utilisé avec succès à travers les années. Cependant, il existe plusieurs interprétations du terme plate-forme [38]. Dans ce mémoire, une plate-forme SoC est généralement composée de modules matériels dédiés et de processeurs capables d'éxecuter des modules fonctionnels exprimés sous la forme logiciels. Ces divers modules sont normalement reliés par un ou plusieurs médiums de communication. Cette architecture, programmable ou fixe est

souvent optimisée pour une classe d'applications. Aujourd'hui, le concept de la réutilisation des composants conçus est la clef fondamentale du succès pour la conception d'une plate-forme SoC. Dans le cadre de cette recherche, nous utiliserons le terme plate-forme pour désigner une architecture comportant des processeurs et des modules matériels avec des bus de communication. La réduction du temps de design est possible grâce à ce concept pour un système sur puce comprenant plusieurs millions de portes logiques.

3.1.2 Intégration

L'intégration est importante pour les plate-formes. Elle consiste à intégrer une collection de modules fonctionnels développés individuellement [38]. L'industrie développe des normes de communication permettant l'uniformisation des interfaces de communication entre les composants. Par exemple, la compagnie Infinite Technologie Corporation propose une intégration «multi-core» pour les systèmes SoC utilisant la norme AMBA [42]. Une méthode de conception est nécessaire pour réaliser l'intégration et la validation du système avec des langages de modélisation. La prochaine section discutera de cette dernière.

3.2 Modélisation hétérogène

La modélisation d'un SoC pour le traitement vidéo demande des outils de conception capables de représenter les limitations et les contraintes de la technologie souhaitée. Dans les systèmes vidéos, les calculs matriciels sont requis pour le traitement en temps réel des images. Le passage de la théorie à l'implémentation demande plusieurs étapes de conception.

3.2.1 Le concept

Lors de la conception d'un système embarqué, il y a deux aspects différents à considérer. Le premier aspect est le développement des modules nécessaires au traitement des données et le deuxième est la méthode de communication entre ces modules. Ce dernier sera traité plus en détail au chapitre suivant. Avant de proposer une méthode de

raffinement progressif entre les langages, il est important de connaître les canaux de communication entre les modules au niveau SoC disponibles à divers niveaux d'abstraction. Ces canaux permettront la communication entre des modules spécifiés par des langages différents.

L'hétérogénéité des SoC requiert une validation par simulation des modules décrits à différents niveaux d'abstraction. L'interconnexion des langages sera possible grâce aux adaptateurs. Ces derniers peuvent transformer des structures pour des besoins spécifiques. La figure 3-1 permet d'expliquer plus facilement ce concept.
Nous pouvons voir les possibilités d'intégration des outils. De façon générale, on veut connecter plusieurs langages ensemble par l'intermédiaire d'adaptateur. Nous définirons trois méthodes pour la réalisation des adaptateurs, soit la méthode directe (par une compilation commune), la méthode par mémoire partagée et la méthode par l'emploi de liens TCP/IP.

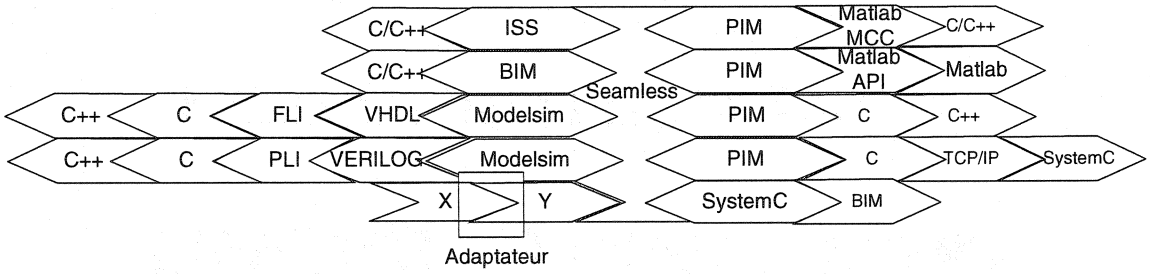


Figure 3-1 Modèle d'interconnexions des langages

Les sections suivantes présentent des adaptateurs spécifiques avec leurs avantages et leurs inconvénients.

L'utilisation d'un environnement hétérogène permet une réduction du temps de développement des systèmes embarqués. En effet, le temps de conception peut être réduit en exploitant divers composants réutilisables exprimés grâce à différents langages et niveaux d'abstraction.

3.2.2 Niveaux d'abstraction

Avant la création d'un modèle hétérogène, il est important de définir quatre niveaux d'abstraction. Chaque module communique à travers des canaux pour échanger des informations avec d'autres composantes du système. En [25], les niveaux d'abstraction sont représentés en fonction du concept. Il y a quatre niveaux de communication lors du raffinement d'un système SoC, soit les niveaux service, transaction, macro-architecture et micro-architecture (RTL). La modélisation hétérogène doit pouvoir supporter tous ces niveaux d'abstraction dans une simulation [25]. Dans ce chapitre, nous utiliserons trois des quatre niveaux d'abstraction, soit les niveaux : transaction, macro-architecture et microarchitecture. L'objectif est de créer une méthode d'interconnexion des langages. Il sera possible d'utiliser les divers niveaux d'abstraction de façon transparente aux modules. C'est-à-dire une communication RTL combinée à une communication transactionnelle de deux simulateurs. La figure 3-2 présente un exemple d'une combinaison de deux niveaux d'abstraction.

Cette première méthode d'interconnexion sera étudiée plus en détail dans les sections suivantes. Dans ce cas, deux simulateurs communiquent par un canal transactionnel.

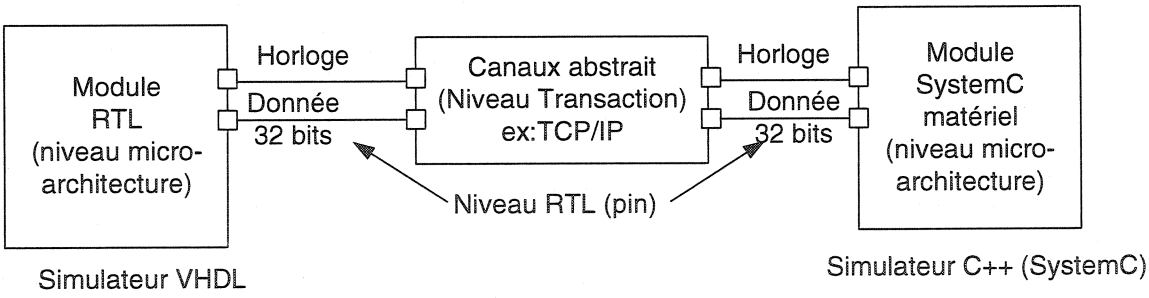


Figure 3-2 Mixage des niveaux d'abstraction

Les informations au niveau broche sont encapsulées pour être transmises et recomposées pour être interprétées par l'autre simulateur. Il est important aussi de connaître les limites des outils et leurs capacités. Les langages principaux avec les niveaux de communication communément utilisés sont décrits en [25]. Les outils disponibles ne supportent généralement pas tous les niveaux d'abstraction pour le raffinement d'un système SoC

[29]. L'accélération du temps de simulation est possible en combinant judicieusement les outils. La capacité de ces derniers a été brièvement décrite dans le chapitre précédent. La section suivante présente trois méthodes générales d'interconnexion applicables à la conception des systèmes embarqués.

3.3 Méthodes d'interconnexion des langages

L'hétérogénéité des composants d'un modèle SoC pour une application requiert plusieurs outils possédant des caractéristiques propres. Dans le cadre de notre recherche sur les outils et leur interaction, nous pouvons définir des méthodes d'intercommunication entre les langages de modélisation. Un langage peut se connecter à un autre langage lorsqu'il possède des caractéristiques communes. La majorité des langages de conception sont capables d'avoir des notions compatibles en C. De plus, certains langages peuvent générer du C ou C++, permettant ainsi une plus grande inter-connectivité dans le cas de langages comme Matlab, UML etc. Donc, le langage doit être capable de se connecter via un API ou de générer du C/C++. Le point commun entre les outils peut être soit le C/C++, une communication réseau ou une plage mémoire partagée (unix).

3.3.1 Structure d'intercommunication

Avant de présenter les méthodes suivantes, il est important de décrire une structure commune à deux langages. La structure proposée et applicable dans les deux dernières méthodes, soit les communications par mémoire partagée sous Unix et les communication par lien TCP/IP (Figure 3-3):

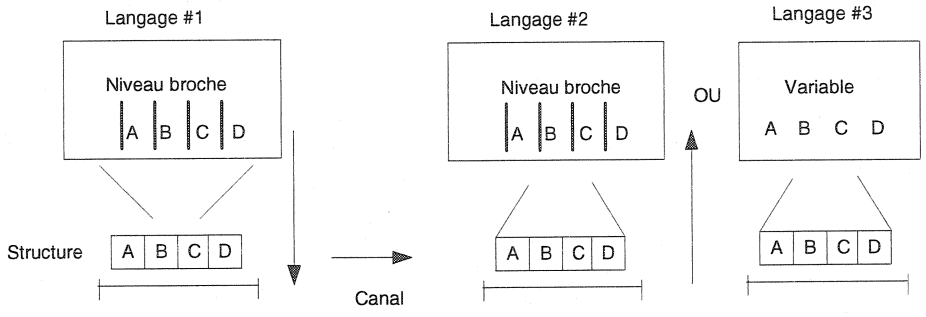


Figure 3-3 Méthode d'échange des structures entre langages

Cette structure permet d'envoyer et de recevoir des données appartenant à des niveaux d'abstraction différents ou identiques dans deux langages. Il est toujours nécessaire d'avoir un adaptateur capable de transformer des structures à divers niveaux d'abstraction pour pouvoir communiquer avec un autre niveau. La structure utilisée sera une *struct* en C possédant des caractéristiques des langages communs.

3.3.2 Méthode standard

La première méthode consiste à incorporer les différentes bibliothèques des langages de modélisation ensemble dans le but de les simuler conjointement. Généralement, les langages ont des bibliothèques disponibles statiquement (Exemple : SystemC) ou dynamiquement (Exemple : Seamless). Les bibliothèques statiques sont compilées avec le code principal. Les bibliothèques dynamiques sont précompilées et liées lors de l'exécution du programme. Le lien entre deux langages peut se faire en déclarant les prototypes des fonctions utilisées dans un programme commun. Il suffit lors de l'exécution qu'il y ait un lien dynamique aux bibliothèques déjà compilées. La figure 3-4 présente la première méthode d'interconnexion des langages.

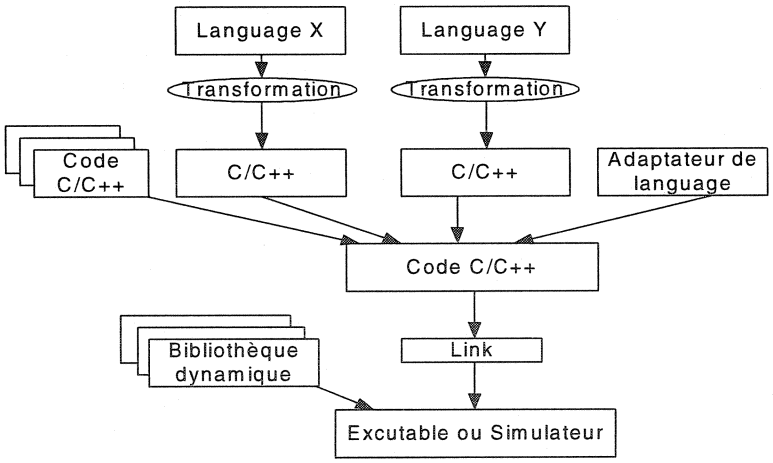


Figure 3-4 Méthode standard de simulation

Un langage de programmation autre que le C/C++ peut posséder des mécanismes de transformation automatique de lui-même en C/C++. Il permet une compilation C/C++ en faisant la liaison des prototypes des fonctions. Lors de l'exécution de la simulation, il y aura l'appel des bibliothèques dynamiques. De plus, il est nécessaire d'avoir un

adaptateur capable de transformer les types de données. Une autre possibilité est l'utilisation d'API en C/C++ des langages permettant une communication extérieure. Ce dernier appelle le langage au lieu d'appeler les bibliothèques.

3.3.3 Méthode de connexion par mémoire partagée UNIX

Une autre méthode d'interconnexion des langages consiste à utiliser la mémoire partagée d'Unix [27]. Le principe est de créer une zone mémoire commune aux deux langages. Dans ce cas-ci, il est nécessaire d'ajouter une variable supplémentaire à la structure pour pouvoir synchroniser les simulateurs ensemble. La figure 3.5 illustre le concept.

Nous attribuons une clef commune permettant de connaître la plage mémoire prise. La variation de la donnée de synchronisation permet de savoir l'état de chaque simulateur. La procédure de codage est similaire à celle de la méthode TCP/IP.

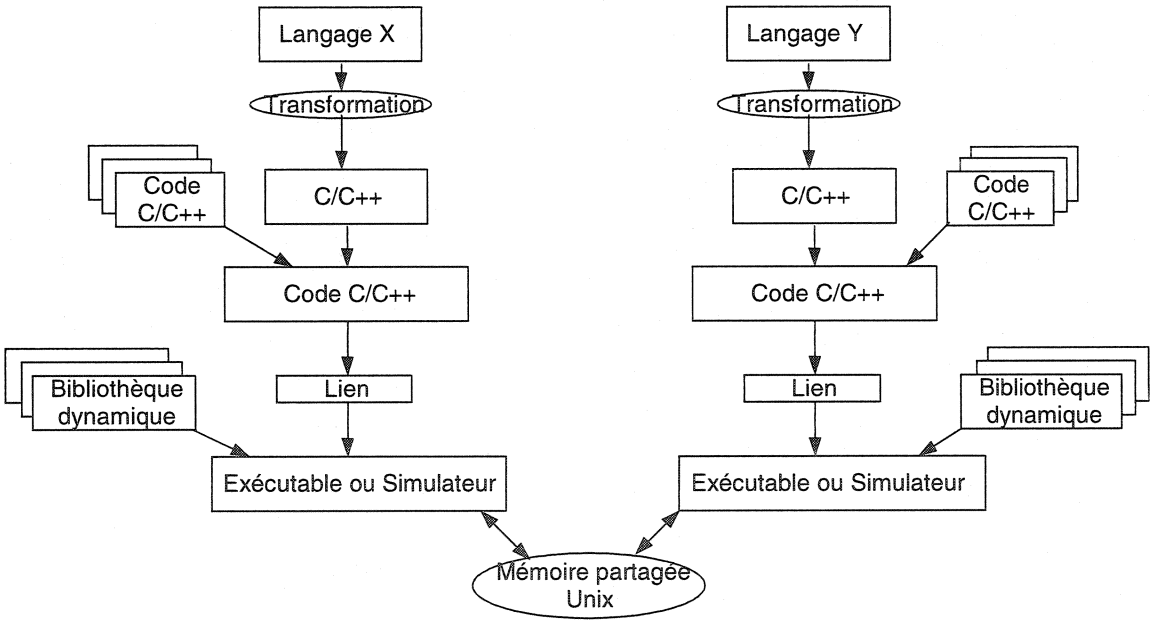


Figure 3-5 Méthode de simulation par mémoire partagée

Les simulateurs doivent initialiser la mémoire partagée. De plus, nous devons définir un serveur et un client pour une simulation hétérogène.

3.3.4 Méthode de connexion par lien TCP/IP

La prochaine méthode consiste à exécuter deux simulateurs en parallèle en excluant les API des langages. La figure 3-6 présente le modèle de simulation. Cette méthode permet l'intégration de deux langages par un canal abstrait. Nous utiliserons la propriété intrinsèque d'une communication TCP/IP [25]. Il y a deux phases importantes lors de la réalisation de ce type d'adaptateur avec la notion d'horloge. Premièrement, il faut trouver la phase d'initialisation des langages visés. Deuxièmement, nous devons trouver le point d'entrée permettant l'avancement des horloges et l'ordonnanceur du simulateur si applicable. Troisièmement, selon les configurations choisies, il est nécessaire de définir le client et le serveur. À l'initialisation, le serveur devra écouter sur le port de communication et le client doit envoyer une demande de connexion.

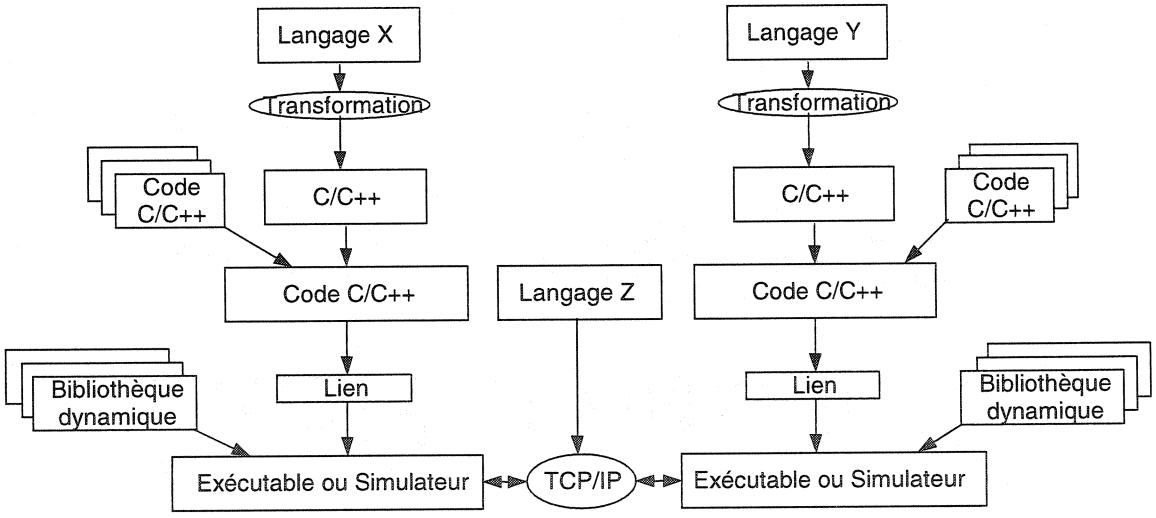


Figure 3-6 Méthode de simulation par lien TCP/IP

Lors de la simulation au niveau horloge, il y a aura des échanges de données entre les simulateurs. Le client écrit une structure telle que présentée précédemment et le serveur le lit pour lui envoyer d'autres informations par la suite.

3.3.5 Comparaison des Méthodes

La simulation de composants homogènes requiert plusieurs mécanismes permettant un environnement de simulation hétérogène des composantes. La connexion demande une certaine compatibilité. Les trois méthodes présentées permettent de réaliser une interface entre les langages.

La première méthode, la plus commune, permet sans aucune structure de communication le lien de deux ou plusieurs simulateurs. Lorsqu'il n'y a pas de possibilité de mettre en commun des bibliothèques statiques ou dynamiques, nous utiliserons les méthodes subséquentes. Il est nécessaire avec la deuxième méthode d'avoir une forme de système d'exploitation Unix ou Linux. L'inconvénient de cette méthode est l'ajout de mécanismes de communication avec la mémoire partagée. L'avantage est de pouvoir transférer en une seule opération de plus grandes quantités de données qu'avec TCP/IP à cause de l'encapsulation des données en paquet TCP/IP. La communication réseau permet par contre de réaliser une simulation sur plusieurs unités de traitement en parallèle. De plus, elle permet de connecter des langages ou outils opérant sous deux systèmes d'exploitation différents. Les sections suivantes présentent des adaptateurs basés sur les méthodes présentées.

3.4 Adaptateurs

3.4.1 Matlab

Le flot conception des traitements vidéos nécessite des algorithmes capables de réaliser la fonction que l'on désire. Il est important d'avoir un langage capable de réaliser facilement des algorithmes dans la méthode. L'intégration de Matlab peut se réaliser de deux façons. La méthode 1 consiste à transformer un fichier Matlab en C/C++. La méthode 2 est l'appel en C de l'application Matlab pour réaliser le traitement. La section suivante comparera les deux méthodes de simulation avec les avantages et inconvénients de chacunes. Il y a deux adaptateurs qui permettent la simulation. Ces derniers sont nécessaires pour faire le lien entre les langages. La fonction adaptateur reçoit un entier et

retourne une valeur. Avec la méthode 2, on ne veut pas démarrer Matlab à chaque requête de l'adaptateur, mais plutôt une seule fois pour lui accéder par la suite. Donc, il est préférable d'exclure l'appel de Matlab dans l'adaptateur pour accélérer le temps de simulation Nous pouvons voir la différence des codes des adaptateurs basés sur l'API de Matlab et celui du code C/C++ de Matlab traduit par l'outil MCC en annexe A. Les deux méthodes contiennent des nombres de lignes comparables en négligeant l'initialisation de l'engin Matlab. L'avantage de ces méthodes est qu'elles permettent l'utilisation de plus de 600 fonctions dans un environnement de simulation [25]. De plus, le code est plus simple en Matlab qu'en C/C++ pour le traitement matriciel. Il y a aussi une possibilité d'exploiter une interface graphique fournie par Matlab lors d'une simulation avec d'autres outils. La méthode 1 permet une simulation plus courte et nécessitant une transformation en C/C++. La méthode 2 permet l'utilisation d'un programme Matlab directement en simulation sans aucune transformation et compilation. La compilation de l'engin Matlab sera requise une seule fois. La méthode 2 permet aussi un raffinement plus rapide du code Matlab dans un environnement hétérogène contenant un SoC. Ce dernier ne nécessite aucune compilation, on peut changer le code de Matlab sans recompiler les adaptateurs et le reste du système. Le tableau 3-1 présente les possibilités entre les deux méthodes.

Tableau 3-1 : Différentes possibilités offertes par les deux méthodes étudiées

	Méthode 1	Méthode 2
M-files ont des espaces de travail	NON	OUI
M-files ont des variables dynamiques	NON	OUI
M-files ont des objets	NON	OUI
Compilation unique	NON	OUI
Raffinement	OUI	OUI
GUI Matlab	OUI	OUI
Autres interconnexions	NON	OUI

Nous pouvons constater que les M-files représentant le code de Matlab ont des limitations d'utilisation avec la compilation automatique de MCC. Cependant, nous pouvons appeler Matlab directement pour avoir plus de fonctionnalité dans un environnement de simulation. La section suivante présente l'interconnexion de SystemC et VHDL pour notre modèle de raffinement progressif.

3.4.2 Liens entre SystemC et VHDL

L'intégration de SystemC au simulateur VHDL par TCP/IP ou par mémoire partagée requiert des adaptateurs capables de synchroniser l'ordonnanceur de SystemC et le noyau de Modelsim. Une communication réseau permet d'exploiter une capacité de calcul parallèle sur plusieurs machines. Tout d'abord, il faut trouver un moyen de synchronisation commun permettant une simulation entre les langages. Ensuite, il faut ajouter une couche réseau TCP/IP pour les communications. Enfin, il faut avoir une structure de données commune aux deux outils pour un dialogue adéquat. La classe SystemC a une grande flexibilité. L'interconnexion de cette classe avec d'autres langages est possible grâce à la gestion par les mécanismes de synchronisation de l'incrémentation des horloges. La première analyse consiste à connaître les méthodes d'avancement d'horloge des deux outils. SystemC permet d'avancer l'horloge de deux façons : soit sous le contrôles de l'usage ou de façon automatique.

La fonction *sc_start(-1)* permet une simulation automatique de l'horloge. C'est-à-dire que l'horloge est gérée par l'ordonnanceur de SystemC. Il y a six étapes [13][41] requise pour effectuer une gestion automatique par SystemC :

- Incrémention des horloges
- Exécution des SC_METHOD/SC_THREAD qui ont un changement à leur entrée.
- Mise à jour des sorties des processeurs de type SC_CTHREAD.
- Si des changements surviennent sur une ou plusieurs sorties, recommencer à partir de 2
- Exécution des SC_CTHREAD. Les sorties seront propagées à l'étape 3 du prochain coup d'horloge.

- Incrémentation du temps de simulation et redémarrage à l'étape initiale.

Lors de la synchronisation entre deux langages, il faut mettre en phase les deux simulateurs. Il est nécessaire d'utiliser la méthode de gestion par l'usager. L'utilisation de la fonction *sc_cycle* de SystemC permet l'avancement de l'horloge. Dans notre cas, nous utilisons le lien TCP/IP ou une mémoire partagée pour une communication entre le FLI [44] de Modelsim et SystemC. Nous choisissons d'utiliser Modelsim comme client et SystemC pour le serveur. Les deux langages ont une fonction de communication TCP/IP permettant de connaître l'état de l'autre simulateur. Les deux fonctions TCP/IP sont la lecture et l'écriture. Nous pouvons distinguer deux phases, soit l'initialisation et l'avancement des horloges. La première phase consiste à se connecter au serveur en TCP/IP et à effectuer un ensemble d'étapes d'initialisation. La seconde phase consiste à créer une synchronisation commune entre les deux langages. La structure envoyée permet l'échange des données entre les simulateurs. Nous devons spécifier le port de communication, le pointeur de la structure et sa taille. Il n'est pas nécessaire d'envoyer les informations de l'horloge. Les fonctions TCP/IP permettant la communication entre les simulateurs servent de façon transparente à faire attendre ou avancer les horloges des langages de modélisation. Modelsim initie une requête et SystemC commence la simulation. Les deux incrémentent l'horloge. Modelsim attendra de recevoir une réponse de SystemC pour continuer la simulation.

L'autre méthode consiste à utiliser la mémoire partagée sous la gestion d'Unix. Il y a trois phases, la création de la mémoire partagée, l'interrogation du simulateur distant et l'échange continuel des données dans le domaine synchrone. Il existe d'autres façons d'intégrer du code C avec VHDL soit par le FLI ou par le PIM. Avec SystemC, nous utilisons une connexion FLI (discutée plus loin). L'utilisation de Seamless permet une communication PIM utilisant de façon transparente le FLI (discutée plus loin) pour la simulation d'un processeur à l'aide d'un simulateur de jeu d'instruction (*instruction set simulator* , ISS). La prochaine section permettra la comparaison des deux langages.

3.4.3 PIM et FLI

Le FLI et le PIM permettent tous les deux de réaliser une simulation conjointe combinant le langage C avec du code VHDL. Le PIM (pin interface module) de Seamless permet une simulation VHDL/C. Dans le cadre d'un développement de projet, il est important de pouvoir simuler tous les niveaux d'abstraction à toutes les fréquences d'opération. La fréquence d'horloge des modules utilisant une connexion PIM est limitée par le ISS. La figure 3-7 présente la limitation du PIM. Le processeur simulé par le ISS permet une simulation des modules en PIM opérant à une fréquence inférieure à celle du CPU.

Une simulation multi fréquentielle à une fréquence plus élevée que celle du ISS est possible en ajoutant une connexion FLI (fig 3.7b). La flèche entre la figure 3.7 a) et b) représente un changement de langage pour permettre une simulation hétérogène. Nous avons programmé une interface en C permettant le lien entre un module d'un autre langage tel que Matlab et SystemC pour qu'il puisse se connecter au bus de contrôle avec le protocole AMBA. Le code PIM de cet adaptateur est en annexe A.

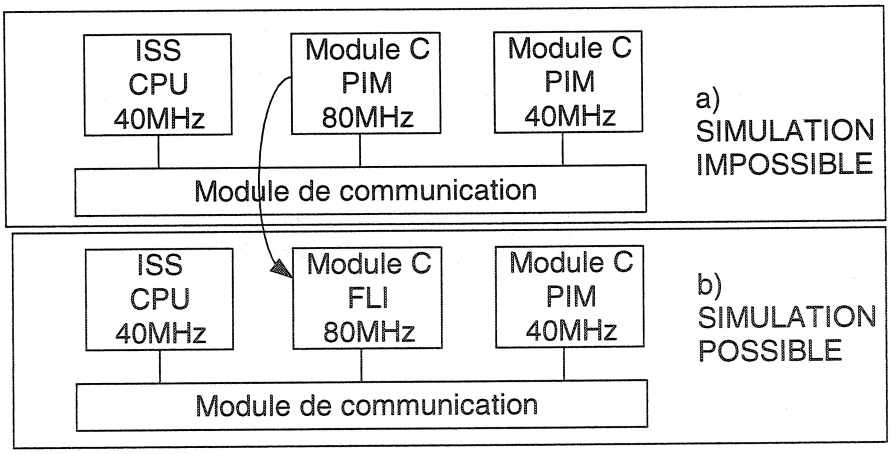


Figure 3-7 Comparaison entre le FLI et le PIM

Nous avons utilisé un processeur ARM. La vérification logicielle et matérielle représentée en C ou autre peut être validée dans une simulation hétérogène. Le modèle de la figure 3.1 du modèle d'interconnexion des langages étant réalisé par l'utilisation des

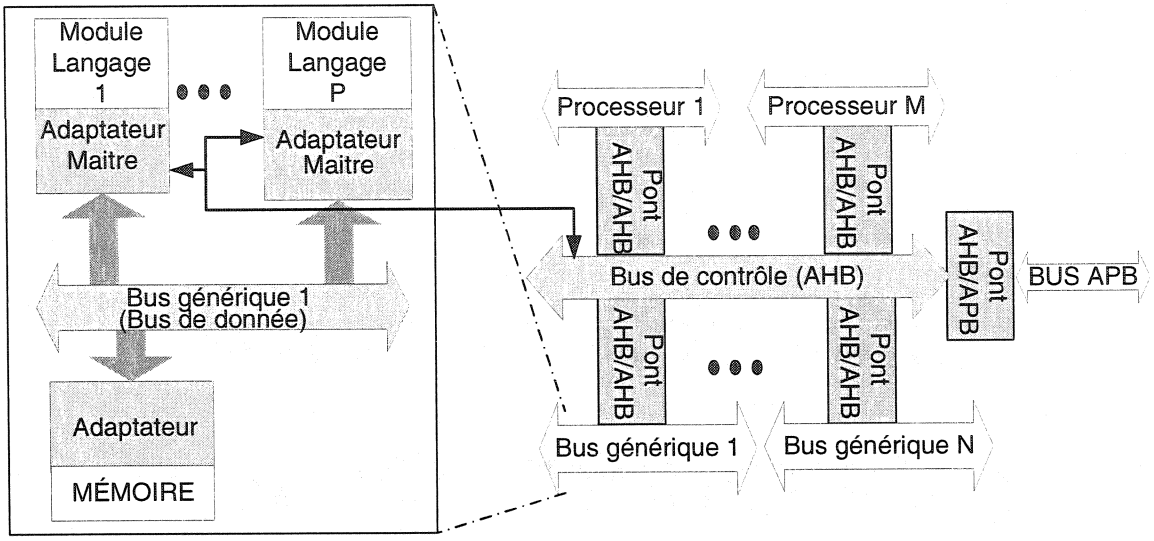
adaptateurs. Il faut maintenant, à partir d'un environnement hétérogène que nous avons conçu et décrit précédemment, définir une plate-forme générale permettant de développer des applications.

3.5 Conception d'une plate-forme SoC

3.5.1 Proposition d'une architecture générale

La clé du succès pour concevoir rapidement une plate-forme SoC est l'utilisation appropriée des langages et leur interconnexion tel que discuté aux sections précédentes. Pour passer d'un algorithme à une plate-forme SoC, on divise la fonctionnalité de celui-ci en modules pour permettre un développement concurrent entre les concepteurs. De plus, nous avons besoin d'une plate-forme SoC générique pour une intégration rapide comme montré à la prochaine figure.

Dans cet exemple, l'architecture est divisée en deux bus communiquant par l'intermédiaire d'un pont. Le bus de droite est un bus de commande et le bus de gauche est un bus de données. On utilise cette distinction dans les plates-formes vidéo comme la Nexperia [5]. Le bus générique de haute performance sera discuté au chapitre suivant. Deux aspects sont importants dans une plate-forme, les modules/sous-systèmes et la communication entre eux.



P: Nombre de langage supporté par l'environnement de simulation
M: Nombre de processeur requis pour l'application
N: Nombre de bus requis pour la bande passante du système

Figure 3-8 Modèle de plate-forme générique

Le système a un processeur consacré au contrôle de la plate-forme et un autre processeur pour le traitement logiciel de l'application si applicable. La communication entre les modules a été définie par le protocole AMBA. L'architecture utilisée ici est basée sur un système de mémoire partagé à un port d'accès émulant un milieu de communication ayant une mémoire à ports multiples. Chaque module peut être développé concurremment parce que les communications entre les modules et la mémoire centrale n'ont aucune latence apparente. La prochaine étape consiste à prendre un algorithme Matlab dans notre cas et à utiliser la propriété de développement concurrent du nouveau bus générique. Pour la suite de la discussion, chaque module sera considéré comme un *thread* ayant la capacité de communiquer avec d'autres *threads* via des mécanismes de communication propre au niveau d'abstraction. De plus, les modules peuvent être disponibles et être simulés dans un modèle hétérogène illustré à la figure 3.1 et qui permet la réutilisation et la réduction du temps de simulation.

3.5.2 Flot de conception

Lors de la conception d'un système de traitement vidéo, il est important de suivre une méthode de conception. La prochaine figure montre étape par étape de façon générale la réalisation d'un SoC partant d'un algorithme Matlab.

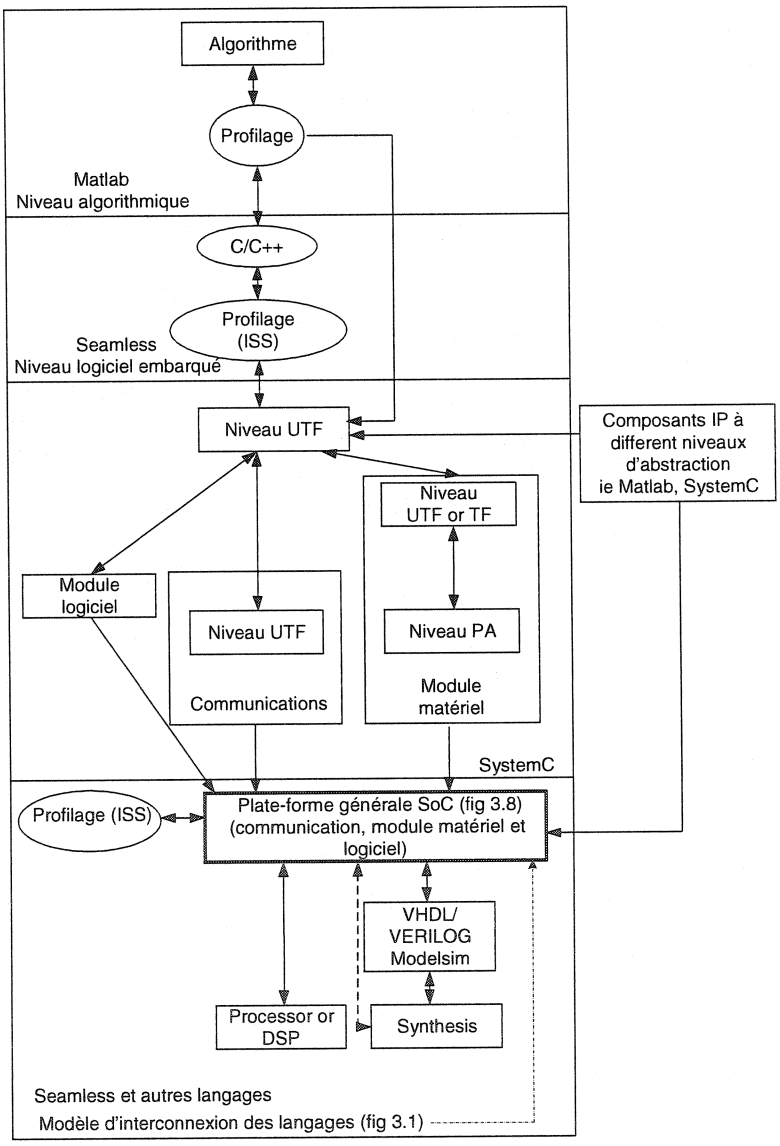


Figure 3-9 Méthode de conception

Cette nouvelle méthode de conception SoC est basée principalement sur Matlab, SystemC et Seamless. Nous avons établi un environnement hétérogène entre ces langues pour simuler tous les niveaux d'abstraction en même temps. De cette façon, le concepteurs peuvent développer leurs modules concurremment à partir de la fonctionnalité de Matlab à la description RTL et/ou logiciel de l'application souhaitée. Avant la division de la fonctionnalité d'un algorithme entre les concepteurs nous avons besoin de plusieurs étapes utilisant deux outils de profilage.

Le premier profilage est employé par Matlab et le second est réalisé par Seamless. Le lien entre Matlab et la description RTL est SystemC. Celui-ci permet l'encapsulation de la fonctionnalité de l'algorithme dans des modules distincts et il représente la communication entre eux. Cette méthode n'exige pas un raffinement de la communication lors de la deuxième réalisation d'une plate-forme SoC, parce qu'elle a été faite avec la conception de la plate-forme de transmission vidéo en utilisant une version d'un bus générique. Le langage Matlab permet le développement d'algorithmes. Dans notre cas, nous utiliserons le filtre de Wiener pour l'illustration de cette méthode. Chaque module sera considéré comme un *thread* en SystemC servant à réaliser un premier raffinement des modules avec une communication par FIFO bloquant. Modelsim servira à la validation de modèles VHDL/VERILOG des modules implémentés au niveau physique. Seamless sera utilisé pour inclure un processeur dans le modèle et il permettra de réaliser un autre niveau de profilage permettant le partitionnement logiciel et matériel d'un processeur donné. De plus, toutes les informations concernant la bande passante et le temps d'attente d'une requête au bus seront disponibles. Les sections suivantes présentent plus de détails à chaque étape de la conception.

3.5.3 Niveau algorithmique

À partir de l'algorithme de Wiener en Matlab, nous pouvons faire un profilage du code illustré à la figure 3.9. Nous employons le code Matlab de plus faible densité pour déterminer comment diviser la fonctionnalité en modules principaux et pour effectuer ensuite la division des tâches entres les concepteurs. Nous avons pris les fonctions ou les

lignes qui demandent le plus de ressources et chacune d'elles est considérée comme un *thread*. Les fonctions qui n'exigent pas beaucoup de temps peuvent être rassemblées dans un module. On peut facilement après cette opération pipeliner les *threads* entre eux selon l'application et les ordonnancer par un processeur contrôlant la plate-forme. Nous avons réalisé un premier niveau d'analyse algorithmique avec le filtre Wiener. Le tableau 3-2 présente un profilage algorithmique du filtre de traitement vidéo d'une image de 65535 pixels.

Tableau 3-2 : Premier niveau de profilage (algorithmique)

Fonction	%Temps	Temps requis (s)*	Description
localMean	43.2	0.9	Calcul de la valeur moyenne d'un bloc dans une image
localVar	36.4	0.160	Calcul de la variance d'un bloc dans une image
Max(I,0)	4.5	0.020	Trouve le maximum de la fonction
Im2double(I)	4.5	0.020	Convertit une image en format double pour Matlab
f=f+localMean	2.3	0.020	Addition de deux vecteurs ensembles
Autres	9.1	0.10	Autres fonctions de l'algorithme
Total	100	0.440	N/A

*Basé sur une fréquence de 400MHz du modèle Matlab

L'analyse permet de voir les fonctions qui demandent le plus de ressources. En partant de ces informations, nous créons les modules principaux. Dans ce cas-ci, les fonctions *localMean* et *localVar* seront deux modules distincts considérés comme des *threads* pouvant être interconnectés sur la plate-forme SoC décrites précédemment. Le profilage de Matlab ne permet pas d'analyser la bande passante entre les modules ni les accès mémoire etc. Il est nécessaire de réaliser un deuxième profilage au niveau d'un processeur embarqué pour connaître d'autres éléments secondaires pouvant être considérés comme des modules ou des fonctions principales. Le premier profilage est un profilage uniquement algorithmique permettant une première division de l'algorithme en modules. Cependant, ce profilage n'est pas approprié pour l'analyse des paramètres de la

plate-forme SoC générique, parce qu'elle utilise d'autres niveaux d'abstraction. Donc, il faut utiliser le deuxième niveau de profilage avec l'outil Seamless permettant entre autre le partitionnement logiciel et matériel. Le but de ce chapitre n'est pas d'effectuer le partitionnement logiciel/matériel, mais plutôt d'effectuer une interconnexion adéquate entre les langages de conception. L'avantage du profilage Matlab au stade algorithmique, c'est de pouvoir diviser les tâches de conception des modules entre les concepteurs sans être obligé d'écrire du code avec un langage qui conduit à de plus grandes densités afin d'avoir une bonne idée des modules à réaliser et de leurs fonctionnalités.

3.5.4 Niveau logiciel embarqué

Dans le cadre de la recherche, nous utiliserons le processeur ARM7 qui était le seul processeur disponible avec l'environnement Seamless au laboratoire lors de notre recherche. Si la fonction souhaitée est exécutée sur un processeur ou un DSP qui répond à nos exigences, notre conception peut se terminer à ce stade. Nous avons pris le filtre de Wiener en le convertissant en C pour l'exécuter sur le ARM. Le deuxième niveau de profilage s'effectue avec une image de 16x16. À ce stade, nous évaluons le temps d'exécution des fonctions et leur dépendance. Nous utilisons le profilage de l'outil Seamless version 5 pour déterminer le temps requis au niveau cycle pour un code C donnée. De plus, l'analyse des dépendances est importante pour savoir si les *threads* (modules) peuvent être développés concurremment et peuvent être orthogonaux. La figure 3-10 présente un résultat similaire à Matlab concernant le profilage de code.

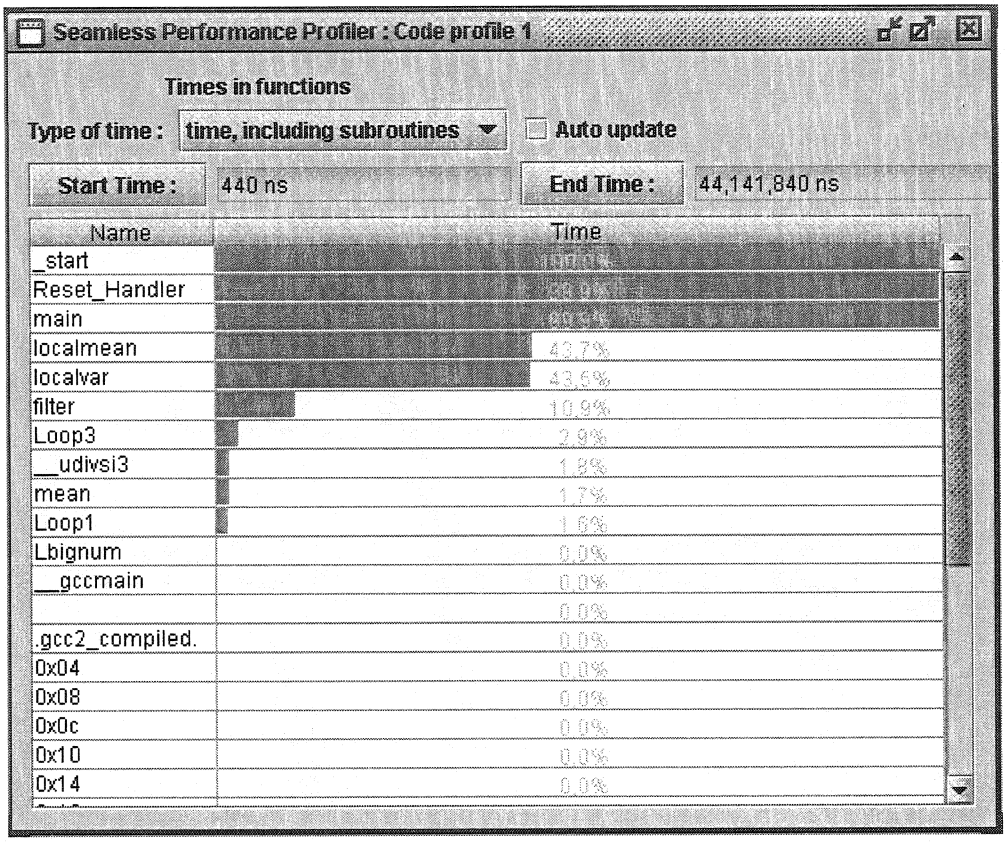


Figure 3-10 Deuxième niveau de profilage (processeur)

On obtient le même type d'information qu'avec le profilage de Matlab avec des éléments supplémentaires plus détaillés. Les modules *localmean* et *localvar* prennent toujours le plus de ressources, soit respectivement 43.7% et 43.5%, en comparaison à 41.25% et 34.76% dans le profilage Matlab. Nous pouvons constater que le profilage algorithmique permet d'estimer approximativement les ressources nécessaires. Nous avons défini à partir de l'algorithme de Matlab deux fonctions additionnelles *mean* et *filter* en regroupant ensemble des lignes de codes qui demandaient moins de ressources. La fonction *filter* (fonction permettant le filtrage de l'image) demande 10.9% des ressources et *mean* (calcul la moyenne) 1.7%. Donc, nous regroupons les fonctions *filter* et *mean* dans un même module. Une autre importante information requise lors de la conception d'un système SoC est le temps requis pour effectuer l'application visée. Dans notre cas, nous avons pris un filtre de Wiener avec une application vidéo HTDV à titre de référence.

Le filtre Wiener est appliqué sur une image. Il y a trente images par seconde de 1920 X 1080 à 24 bits par pixels qui requièrent une bande passante de 1.49 Gps [45]. Le tableau 3-3 présente le temps d'exécution requis pour chaque fonction extrait à partir d'un diagramme de Gant fourni par Seamless.

Tableau 3-3 : Deuxième niveau de profilage (processeur)

Fonction	Temps requis pour 256 points.(ms)
Localmean	19.304
Localvar	19.14
Mean	0.66
Filter	4.95
Total	44.054

Dans notre cas, 256 points prennent 44ms, le traitement de ce type de données prend au moins 352 secondes. Ce qui n'est pas acceptable. Ce sont des fonctions qui réalisent des traitements vectoriels et qui nécessitent des traitements en boucle. Nous avons décidé de réaliser les blocs en matériel dû à la disponibilité uniquement des processeurs ARM7 au laboratoire. De plus, la réalisation en matériel de ces fonctions permet un haut niveau de parallélisme. Nous avons défini 4 modules principaux à partir des résultats de profilage. Pour la création des modules et d'une interconnexion entre ceux-ci, il est nécessaire de partir au niveau UTF en SystemC en conservant les informations recueillies précédemment.

Pour éviter de créer un ordonnanceur sur le processeur de contrôle d'une plate-forme SoC à ce stade de la conception, il est nécessaire de passer en SystemC avec une communication basée sur un FIFO bloquant pour représenter un module matériel ou logiciel. Les fonctions représentant des *threads* de calculs seront encapsulées sous forme de modules ayant chacun une fonctionnalité propre. Ces modules sont reliés par des ports de communication. À ce niveau, nous avons l'algorithme Matlab et le code C correspondant. Il est maintenant nécessaire de faire la transposition de la fonctionnalité

globale au travers plusieurs modules décrits en SystemC. La prochaine section présente ce passage.

3.5.5 Niveau UTF (untimed functional)

Nous sommes rendus à la section qui porte sur le raffinement en SystemC de notre méthode de conception tel qu'illustré à la figure 3-9. Le niveau UTF permet de réaliser une simulation sans notion de temps. À partir du logiciel embarqué décrit dans la section précédente, nous pouvons créer un prototype en SystemC permettant d'avoir des modules distincts dans le système. La figure 3.11 montre l'ébauche de la première structure :

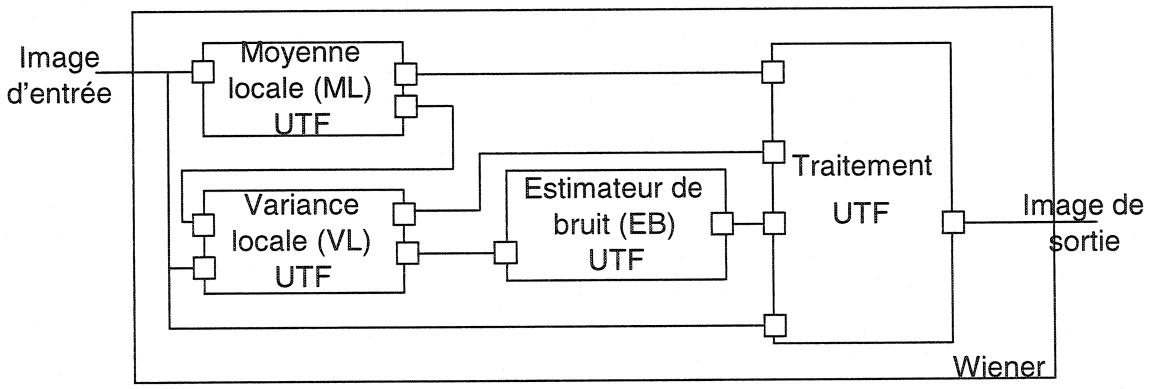


Figure 3-11 Niveau UTF de l'algorithme de Wiener

En partant des étapes antérieures, nous avons divisé le programme Matlab en 4 modules distincts déterminés par le profilage. Ceci nous a permis de déterminer leurs fonctionnalités. ML et VL (voir Fig. 3-11) ont été définis à partir du premier niveau de profilage comme mentionné précédemment. L'estimateur de bruit et le module de traitement (*filter*) ont été définis à partir du code Matlab et confirmés dans le deuxième niveau de profilage. Il faut noter que le module de traitement et l'estimateur de bruit peuvent être mis ensemble. Dans notre cas, nous les avons séparé pour avoir plus de modularité de conception (un module par concepteur). Chaque ligne de l'image précédente représente des canaux abstraits de communication. Ces canaux sont des FIFO bloquant possédant des fonctions de lecture et d'écriture. Nous parlons alors de communications au niveau TLM (Transaction Level Model) pour lesquelles des normes

basées sur SystemC ont récemment été proposées [18]. Les *threads* sont bloqués par des FIFO bloquant à leur entrée. Ils fournissent leur réponse par un autre FIFO bloquant. Par conséquent, la fonction du module sera exécutée quand les éléments nécessaires seront présent. À ce stade de la conception, nous avons défini une plate-forme générale et des mécanismes pour déterminer la répartition des modules sur la plate-forme générique illustrée à la figure 3-8. Nous transposerons les FIFO bloquant dans le modèle de la plate-forme générique où ces communications sont en fait effectuées au travers une mémoire partagée. C'est donc dire que la commande pour chercher les données sur le bus et pour commencer le traitement sera dicté par le processeur sur le bus de contrôle. Nous faisons l'analogie entre les FIFO bloquant et l'ordonnancement futur des modules de la plate-forme. Les modules récupèrent les données en mémoire par le bus de données pour réaliser des fonctions sur les données et ensuite les résultats sont mis en mémoire pour les passer à un autre module. Chaque canal représente deux liens à un mémoire ou un lien direct entre deux modules.

À ce niveau d'abstraction, il est possible de combiner deux modèles UTF décrits avec des langages différents, soit SystemC et Matlab, en utilisant des adaptateurs. La figure 3-12 présente le début d'un raffinement de modules de réducteur de bruit vidéo partant de Matlab. La connexion choisie est l'API de Matlab avec un adaptateur présenté antérieurement. Nous partons d'un traitement matriciel et nous pouvons raffiner la fonctionnalité du réducteur de bruit vidéo en Matlab en conservant une communication TLM en SystemC.

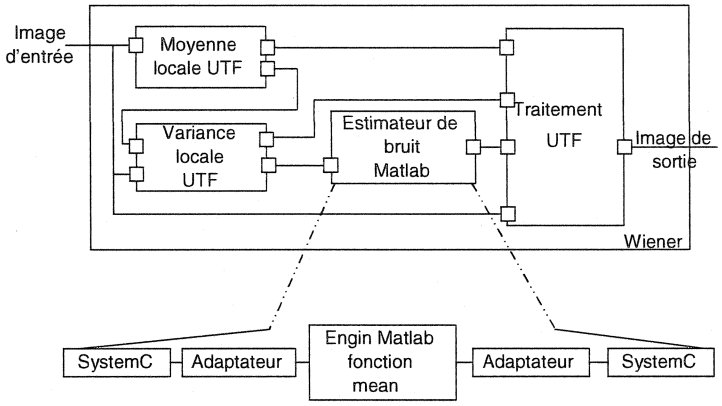


Figure 3-12 Représentation de niveau UTF de l'algorithme de Wiener combinant des modèles Matlab/SystemC

Nous pouvons simuler avec Matlab et SystemC pour avoir la fonctionnalité que l'on désire pour chaque module. Ceci nous donne la possibilité de garder toute la fonctionnalité de l'algorithme et de vérifier les résultats produits sans en avoir raffiné tous les détails. Une fois les critères de fonctionnalité rencontrés, nous passons à l'étape suivante. Un autre avantage qui découle de l'utilisation de Matlab est la grande densité du code par rapport à SystemC tel qu'illustré à la figure 3.23.

3.5.6 Niveau Architectural

À partir de l'architecture générale, nous pouvons définir le système de traitement vidéo au niveau plate-forme. La figure 3-13 présente un ensemble de modules envisagés reliés par des les liens de communication. Comme mentionné précédemment, les modules en SystemC sont les modules de la plate-forme reliés par des canaux abstraits qui sont des liens sur le bus de données. Les modules ont été réalisés en SystemC et les communications en VHDL. La moyenne locale et la variance locale ont un lien commun, car il n'était pas nécessaire de réécrire l'information en mémoire parce que le résultat du premier est nécessaire au résultat du deuxième. Par contre, les deux sont connectés sur le bus de données.

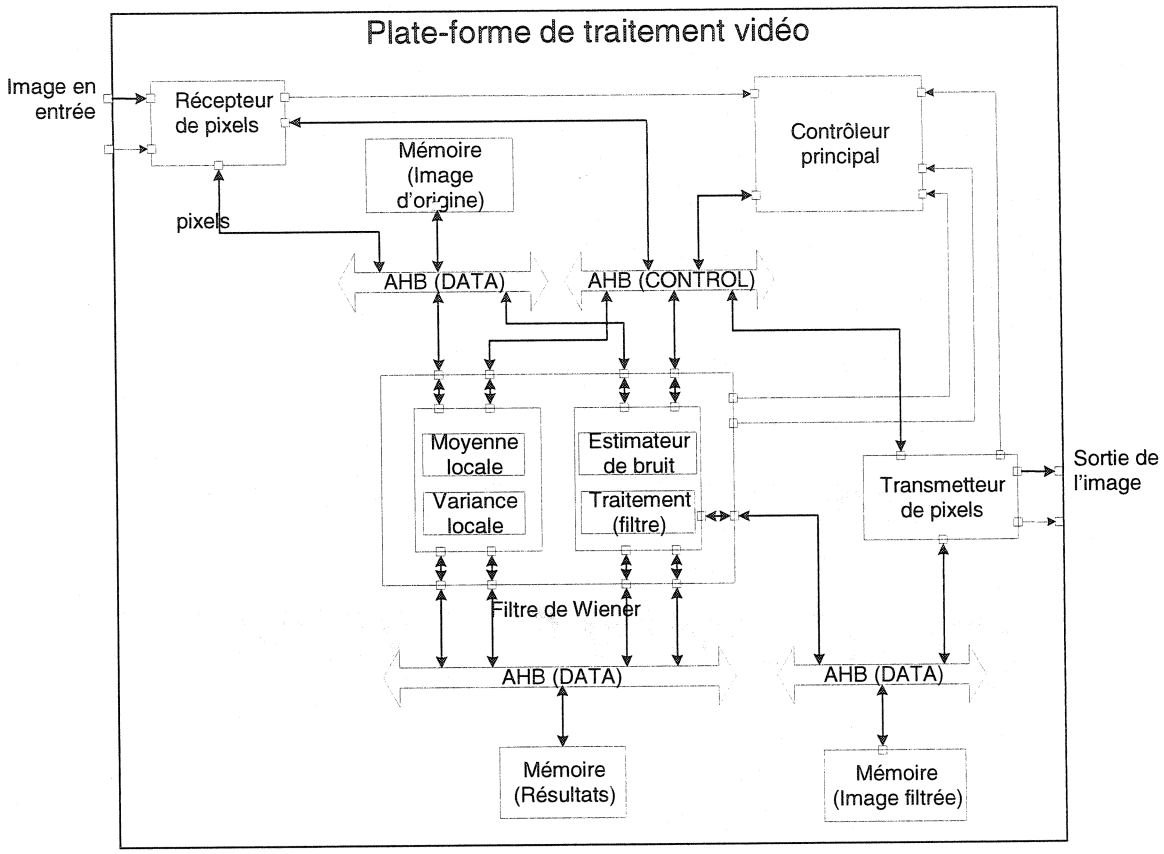


Figure 3-13 Architecture d'une plate-forme vidéo

Dans cette plate-forme, nous retrouvons les quatre modules nécessaires pour le traitement vidéo. Il y a le contrôleur permettant d'assurer la cohérence des données qui est responsable d'ordonnancer les tâches du système. De plus, on ajoute des modules servant à la transmission et à la réception de trames vidéo. Le raffinement des communications sera décrit ultérieurement. Nous utilisons le maximum de la capacité du bus générique de communication pour obtenir la bande passante nécessaire.

3.5.7 Raffinement des modules

Avant l'interconnexion des modules sur la plate-forme, il faut s'assurer que les modules font leurs tâches au niveau d'abstraction le plus bas. Donc, nous avons créé un banc d'essai pour chaque module principal en le conservant jusqu'à la validation complète de celui-ci. Nous avons appliqué cette méthode pour le module estimateur de bruit. La figure 3-14 est une première proposition pour la structure interne de ce module.

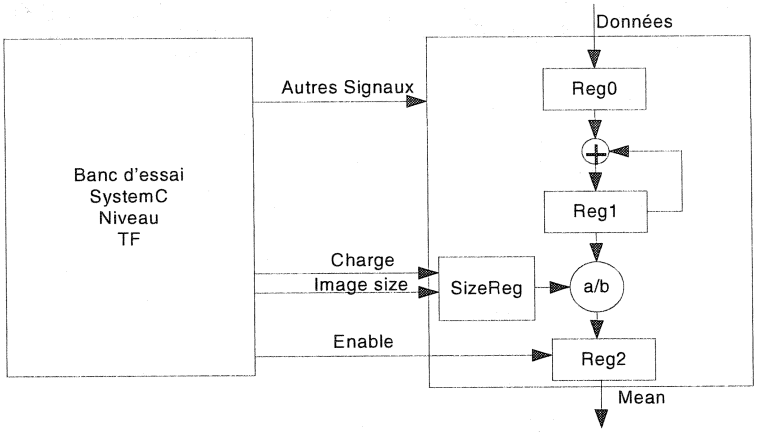


Figure 3-14 Modèle de niveau PA de réducteur de bruit vidéo

Nous simulons ce modèle en supposant que la division se fait en un cycle d'horloge. Le module en SystemC sera exprimé au niveau *pin accurate* (PA) et il sera vérifié grâce à un banc d'essai au niveau *timed functional* (TF). Par la suite, il faut passer au niveau RTL.

À ce niveau, nous avons créé le module estimateur de bruit à l'aide d'IP disponibles. Nous avons pris un diviseur sur le site [33] et créé le reste du modèle à partir de la figure précédente. Nous avons constaté que la division prend plus qu'un cycle d'horloge. Elle a une latence de neuf cycles pour la première donnée tel qu'illustré à la figure 3.17. Nous avons réalisé une synthèse du module pour connaître la fréquence maximale de ce dernier. La vérification a été faite en gardant le même banc d'essai en SystemC au niveau TF qui avait été utilisé aux niveaux discutés plus tôt. Pour notre réducteur de bruit vidéo en utilisant l'outil de synthèse de Synopsis, nous avons une fréquence d'opération d'environ 21.9MHz avec une surface de 53930 unités. Il est important de mentionner que notre but est d'utiliser le plus possible des IP disponibles pour réduire le temps de conception. Si aucun IP ne correspond à nos besoins, nous devons le concevoir. Une fois les caractéristiques rencontrées, nous gardons le niveau PA pour le raffinement des autres modules. Il est alors possible de remonter à un niveau d'abstraction plus élevé pour une simulation plus rapide. La figure 3.15 présente le raffinement d'un module à partir du niveau PA. Tandis que la figure 3.16 présente un sous module RD (registre à décalage)

ajouté à l'estimateur de bruit vidéo. Ce dernier permet d'équilibrer la sortie avec le délai de la division.

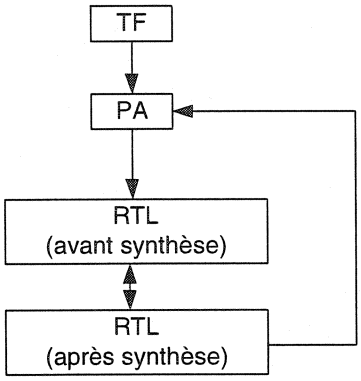


Figure 3-15 Raffinement a un plus bas niveau d'abstraction

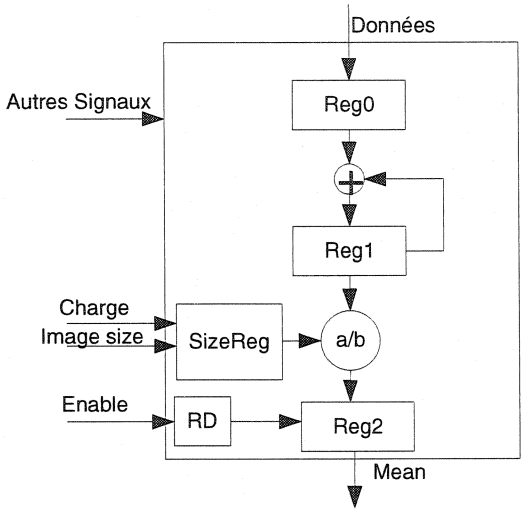


Figure 3-16 Ajustement du module de l'estimateur à partir des composants disponibles

---

Lorsqu'on veut continuer la conception, nous devons ajouter au niveau d'abstraction précédant le délai réel que l'on obtient. De plus, nous obtenons d'autres informations plus détaillées après synthèse pour avoir une estimation de la performance du module et de sa surface dans la technologie souhaitée.

Nous avons raffiné le module d'estimation de bruit de l'algorithme Matlab à SystemC et par suite SystemC à RTL. Les trois figures (3-17 à 3-19) montrent les simulations pour SystemC à RTL entre les divers niveaux pour obtenir la représentation de la figure 3.17. La première figure illustre une simulation RTL, la deuxième une simulation systemC au niveau PA et pour terminer, nous présentons à titre indicatif une démonstration de la conservation de la latence au niveau d'une simulation SystemC/VHDL des délais au niveau TF en combinaison avec le VHDL (non applicable pour le module d'estimateur de bruit).

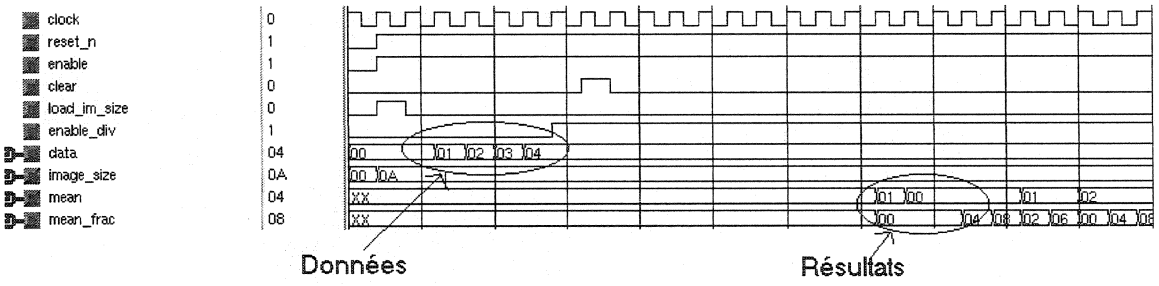


Figure 3-17 Simulation modelsim (VHDL/Verilog)

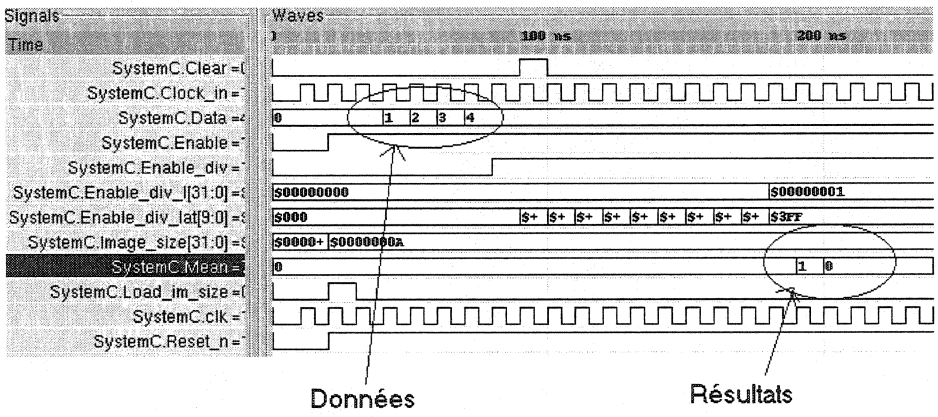


Figure 3-18 Simulation SystemC

Nous pouvons voir les résultats de simulation obtenus pour un estimateur de bruit de base qui réalise la moyenne des nombres à son entrée à chaque cycle d'horloge avec des signaux de contrôle permettant de vérifier la cohérence des données voulues. Pour notre exemple, la somme des nombres va de un à quatre et est divisée par 10, ce qui donne un résultat de 1.

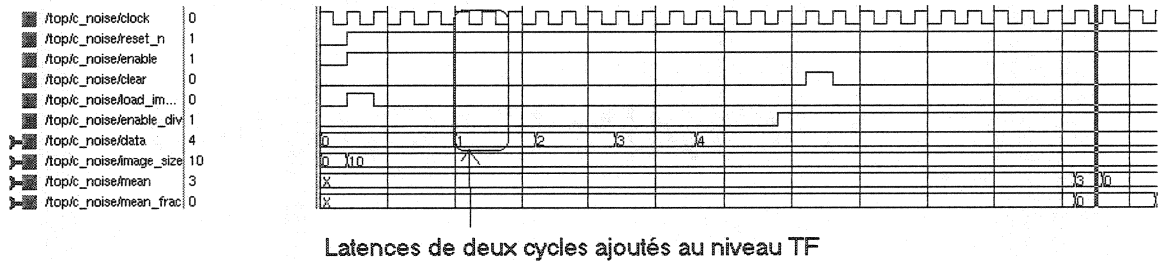


Figure 3-19 Illustration de l'ajout du délai dans une simulation SystemC/VHDL

Lorsque tous les modules sont raffinés, il reste juste le logiciel et le matériel. En résumé, la figure 3-20 montre le cheminement possible d'une fonction Matlab.

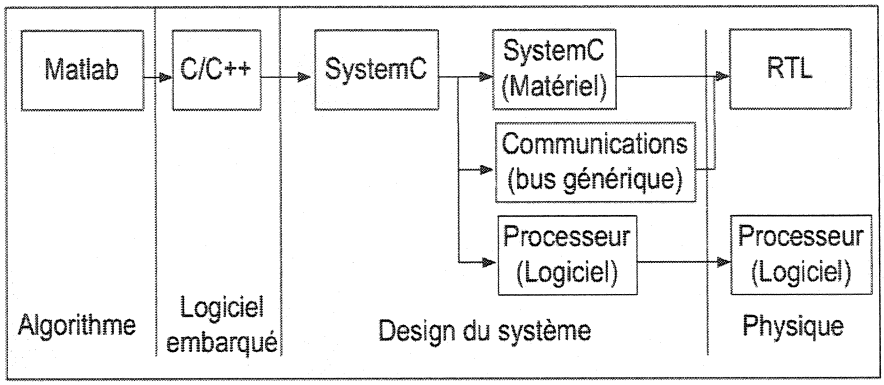


Figure 3-20 Étape de raffinement d'un module

La fonction peut être en matériel ou en logiciel selon l'application et les contraintes de performance. Dans notre cas, nous avons mis les modules en matériel. Également important, il est nécessaire de faire un raffinement des communications tel que décrit à la section suivante. Tous les éléments matériels ou les processeurs du système possèdent des mécanismes de communication selon la norme AMBA pour leur interaction entre eux.

3.5.8 Raffinement des communications

La communication entre les modules est possible par le bus générique AMBA correspondant a un bus de donnée.. Le bus de contrôle de la plate-forme est décrit à la section suivante. Nous avons utilisé un modèle en SystemC pour émuler son comportement. Ensuite, nous avons créé le bus générique en VHDL pour connaître ses performances. Les variantes de ce dernier seront décrites dans le chapitre suivant. Les nombres de bus génériques et de mémoires nécessaires dépendent de leurs capacités. Quand la capacité maximale du bus ou de la mémoire est atteinte, nous créons un autre bus avec une autre mémoire tout en conservant la cohérence des données entre les modules.

3.5.9 Co-vérification et bus de contrôle

Les modules de la plate-forme sont conçus avec une interface esclave pour le bus de contrôle et une interface maître pour le bus de données. Nous avons discuté d'un processeur permettant le contrôle de la plate-forme. Cette section explique brièvement

les mécanismes d'interconnexion des modules et d'un processeur ARM communicant avec les modules. Une simulation matérielle et logicielle de l'architecture générale proposée sera possible avec Seamless. Nous avons pris le cas spécifique de la plate-forme de conversion de protocoles. Dans ce cas, nous avons un bus de contrôle opérant à 40 MHz et un bus de données opérant à une fréquence de 320MHz. De plus, nous avons validé une simulation hétérogène en remplaçant les modules matériels dans des niveaux d'abstraction plus élevés par des modèles Matlab et SystemC comme illustré avec l'architecture générale. Nous avons créé plusieurs émulateurs maîtres et esclaves ainsi que leurs adaptateurs AMBA pour le bus de contrôle et de données. Les émulateurs permettent d'évaluer les performances de l'architecture. Les émulateurs maîtres possèdent une donnée et une adresse propres pour effectuer des accès mémoires. Les émulateurs esclaves sont plus simples que les émulateurs maîtres. Les données écrites dans cet émulateur sont conservées dans une mémoire interne et elles sont renvoyées lors d'une lecture. La figure ci-dessous illustre la connexion entre deux bus au travers d'un pont AHB/AHB.

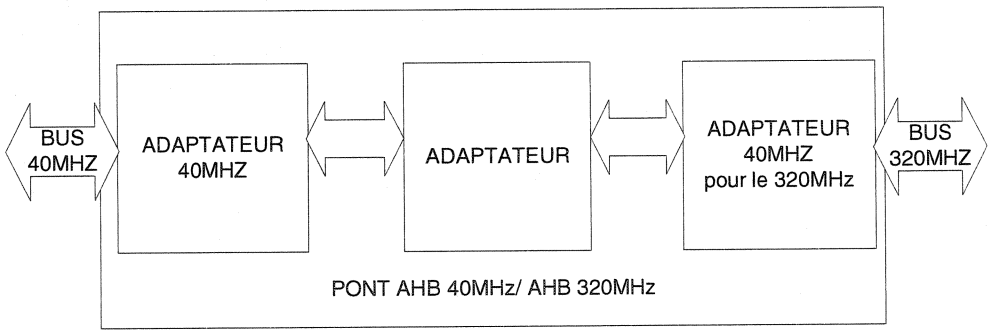


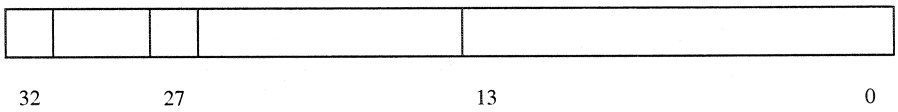
Figure 3-21 Pont AHB 40MHz/ AHB 320MHz

Le bus de données doit être connecté au bus de contrôle par le biais d'un pont. Pour concevoir ce pont, il suffit de jumeler un synchroniseur 320/40 MHz avec un adaptateur 40MHz. Le pont joue un rôle d'esclave sur le bus 40MHz et un rôle de maître sur le bus 320 MHz, ce qui permet à un maître du bus de contrôle (un processeur ARM) d'effectuer une requête sur un esclave du bus de donnée (la mémoire en l'occurrence).

Les requêtes de lecture et d'écriture en mémoire provenant du ARM circulent du bus 40MHz vers le bus 320 MHz à travers le pont.

L'espace d'adressage est défini comme suit :

- Adresses sur 32 bits



- Le bit 31 désigne l'espace d'adresse du bus 40 MHz.
- Le bit 27 désigne le pont vers le bus 320MHz.
- Les 14 derniers bits servent à préciser l'adresse mémoire.

Un banc d'essai simple a permis de valider le fonctionnement du pont en utilisant le caractère 'C' pour écrire et lire en mémoire.

La vérification du bus de haute performance sera discutée dans le chapitre suivant. Pour la validation des modules matériels, les lettres A à N ont été écrites par un processeur ARM, ce qui correspond à tester les connexions esclaves. Ensuite celui-ci relit l'information pour l'afficher à l'écran. La figure 3-22 présente le résultat.

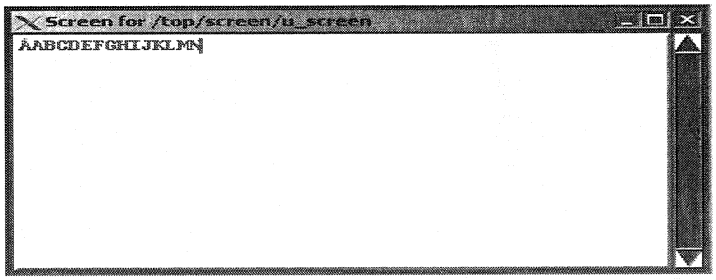


Figure 3-22 Affichage de l'écran avec le bus de contrôle 40MHz

Il faut noter que le premier caractère affiché par l'écran est invalide, celui-ci étant créé lors de l'initialisation du module Screen dans Seamless. Nous avons créé le modèle de la plate-forme en utilisant principalement Matlab, SystemC et Seamless.

3.6 L'utilisation adéquate des langages

D'après notre expérimentation, la figure 3-23 présente les temps de simulation et la densité du code lorsque les divers langages considérés sont utilisés pour réaliser un réducteur de bruit sur la plate-forme de traitement vidéo. La fonctionnalité est conservée du haut niveau jusqu'au bas niveau.

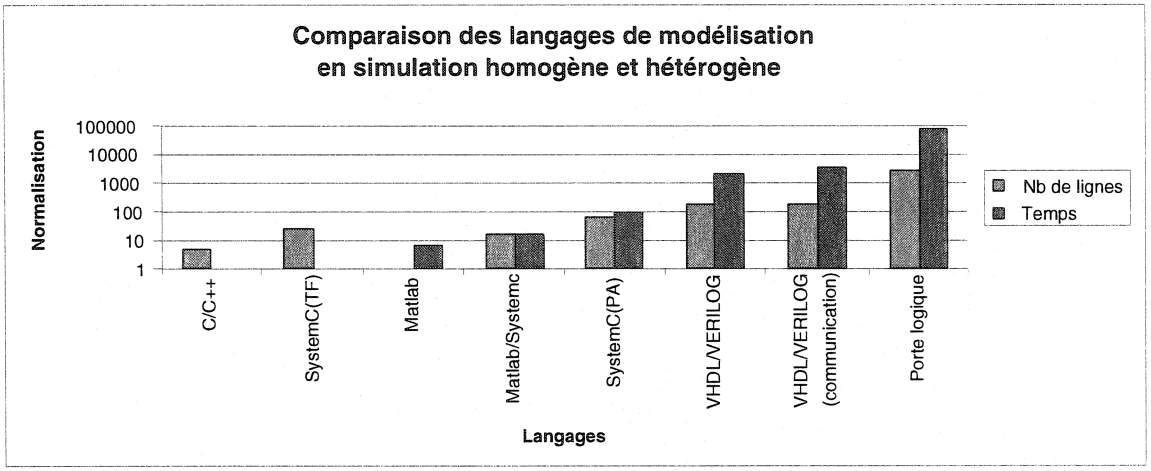


Figure 3-23 Comparaison de l'hétérogénéité des langages

D'après ces informations, nous pouvons utiliser la force de chaque langage de conception pour créer un environnement de simulatiwon commun. Dans la figure précédente, nous utilisons un banc d'essai en SystemC pour les niveaux VHDL et Verilog. Nous pouvons constater de façon générale qu'il y a une pénalité lors de la connexion entre deux langages. De la même façon, nous avons une pénalité du temps d'exécution avec Matlab et SystemC, mais il y a un gain au niveau de la densité du code. Matlab ayant la plus grande densité de code, il servira pour une validation de la fonctionnalité requise d'un module. Seamless et Matlab permettent de définir les modules principaux à mettre sur le bus de communication générique à l'aide de leurs outils d'analyse. Il est possible de prendre plusieurs lignes de code en Matlab et d'en faire un module distinct en conservant l'intégrité de la fonctionnalité visée. De plus, Seamless permet une co-vérification matérielle et logicielle pour les processeurs et DSP. SystemC au niveau TF/UTF permet le design d'un système dans un niveau d'abstraction plus élevé. Une fois que la fonctionnalité des modules en Matlab correspond à nos exigences, les modules matériels

seront modélisés en SystemC au niveau TF/UTF et par la suite au niveau PA. Chaque module doit ensuite être évalué au niveau RTL pour en estimer avec précision la vitesse, la complexité, la puissance, la surface etc. Si les performances ne sont pas satisfaisantes, nous ajustons alors les modèles au niveau TF ou PA. Le SystemC exprimé au niveau PA permet une simulation plus rapide que le modèle au niveau RTL, mais il est nécessaire d'aller à ce niveau pour l'implémentation physique et la caractérisation précise des performances. Le tableau 3-4 résume les principales informations que l'on retire de chaque langage ou outil sur la base de notre expérience.

Tableau 3-4 : Informations récapitulatif de langages

	Langages/outils	Principales Informations
Analyse plate-forme avec DSP et CPU	Seamless	Bande passante, profilage du code, Diagramme Gant, accès-mémoire, etc.
Algorithmique	Matlab	Profilage de code
Logiciel embarqué	C/C++	Densité du code, modularité des fonctions
Design du système	SystemC	Fonctionnalité des modules, délai, ports de communication, etc.
RTL avant synthèse	VHDL/VERILOG	Fonctionnalité de l'implémentation
RTL après synthèse	VHDL/VERILOG	Surface, vitesse, puissance

La partie logicielle, si nécessaire, sera directement mise dans un processeur ou un DSP selon l'application. De plus, nous pouvons constater qu'il y a une pénalité lors de l'interconnexion des langages. L'API de Matlab permet le raffinement plus rapide de la fonctionnalité visée nécessitant une compilation unique avec le reste du système en SystemC. La figure 3-24 présente les langages en fonction de leur capacité à partir de la littérature du chapitre précédant et de nos résultats de recherche.

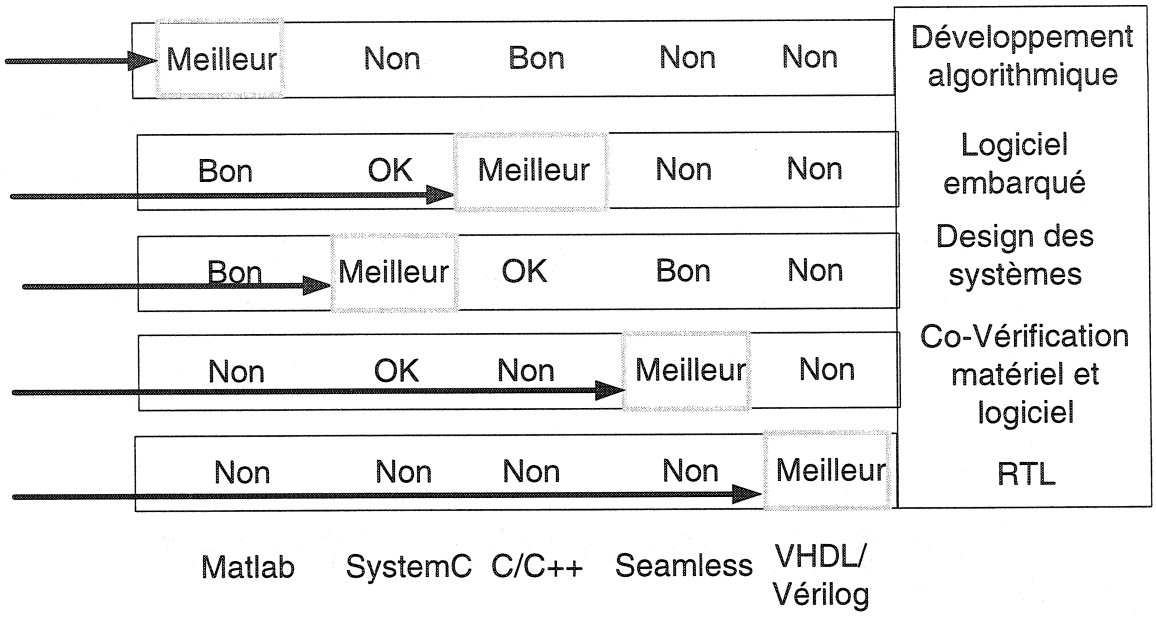


Figure 3-24 Comparaison des langages avec leur capacités

en se basant sur une revue de littérature et notre expérimentation

En résumé, notre méthode de conception hétérogène utilise le meilleur de chaque langage à divers niveaux d'abstraction. Elle permet une conception modulaire déterminée par un double profilage. Elle permet aussi de répartir le travail entre les concepteurs en se basant sur une plate-forme de référence et un système d'interconnexion de haute performance.

3.7 Conclusion

Nous avons réalisé une plate-forme vidéo à l'aide de plusieurs langages de conception avec une simulation hétérogène. Les résultats montrent qu'il est préférable de travailler autant que possible avec les langages de haut niveau d'abstraction, car ils demandent moins de temps de simulation. Lors d'un raffinement, on conserve la structure du système en utilisant les adaptateurs appropriés qui permettent d'interfacer les modèles exprimés à des niveaux d'abstraction distincts. Les bancs d'essai et les modèles comportementaux peuvent être réutilisés à divers niveaux d'abstraction. On choisit l'outil avec la plus grande densité de code pour une exploration de la fonctionnalité des modules

et SystemC pour une représentation du matériel. Seamless servira pour la co-vérification matérielle et logicielle. La méthode à double profilage permet l'utilisation adéquate des langages pour la conception des plate-formes SoC. Le profilage de Matlab permet la division des modules au stade algorithmique, tandis que celui avec Seamless permet un raffinement plus précis et plus spécialisé pour la plate-forme de référence. Un fois que le bus de communication générique a été développé et validé, nous pouvons faire un raffinement des modules à travers les niveaux d'abstraction tout en conversant la fonctionnalité du système.

CHAPITRE 4
BUS GÉNÉRIQUE DE HAUTE VITESSE

Relier l'ensemble des modules d'une plate-forme SoC exige qu'ils soient compatibles. Plusieurs solutions sont disponibles, mais elles manquent de bande passante ou de flexibilité. Ce travail propose des architectures d'interconnexion qui permettent une communication performante entre les modules des systèmes sur puce, tout en leur fournissant une largeur de bande variable. Ces architectures sont basées sur le protocole AHB d'AMBA. Le modèle proposé pour obtenir un système multi-fréquentiel basé sur la norme AMBA est de mettre un encodeur de priorité combiné à l'adaptateur générique dans un ordre cohérent. Les requêtes de l'adaptateur générique sont acheminées vers l'arbitre du système. Il est important d'avoir un ordre par lequel les modules seront traités. La plus haute priorité sera le module opérant à la plus grande fréquence. Si tous les modules opèrent à la même fréquence, nous sommes dans un cas spécifique du modèle proposé, les modules peuvent s'exécuter dans n'importe quel ordre. Nous retrouvons alors un système ne nécessitant pas d'adaptateur AMBA. Lorsqu'il y a deux modules opérants à la même fréquence et d'autres modules à des fréquences inférieures, le choix de l'ordre est à la discrétion du concepteur. Il est important de mentionner que l'accès au bus AMBA demandé par l'adaptateur change à tous les cycles et nous utilisons le mode de transferts simple de la norme AMBA. Pour un système sans latence, il faut toujours conserver une bande passante du système inférieure ou égale à celle de la somme des fréquences des modules. Pour bien comprendre notre modèle, il nous faut d'abord présenter deux autres versions antérieures du bus générique pour en arriver à ce que l'on recherche. L'architecture proposée a été implémentée dans un environnement graphique qui utilise une bibliothèque de cellules développée pour la technologie CMOS 0.18 µm de la société TSMC.

4.1 Modèle d'interconnexion

4.1.1 Méthode de conception

La clé du succès avec les plate-formes SoC de haute performance est de savoir comment relier les modules afin de supporter des transferts efficaces de grandes quantités de données. Les questions clés avec les SoC sont la flexibilité et la capacité d'échanger sur demande des données à haut débit entre les modules du système. L'architecture explorée est basée sur un système ou les communications se produisent au travers une mémoire partagée émulant un mécanisme de communication basé sur une mémoire à plusieurs ports.

Les SoCs sont souvent optimisés en séparant certains de leurs sous-systèmes importants tels que les processeurs, les mémoires et les entrées/sorties (E/S). Un autre problème important est comment relier ensemble ces modules efficacement. Un but important est de fournir une intégration sans goulot d'étranglement pour des communications entre les modules et les transferts d'E/S. Également important, la solution d'intercommunication devrait être flexible pour faciliter l'intégration d'une large classe d'applications. Plusieurs méthodes sont disponibles pour relier ensemble des modules dans des plate-formes SoC. La méthode proposée exploite et constitue une valeur ajoutée à la norme AMBA-AHB (Advanced High-Speed Bus) [8] pour implémenter une infrastructure flexible de communication de haute performance pour les SoC. AMBA est typiquement utilisé dans les systèmes basés sur les processeurs ARM. Dans ces systèmes, AMBA est un module de communication pour un processeur. Il fournit une largeur de bande limitée et mène à des latences significatives. Toutefois, ce travail montrera que la norme AMBA peut mener à des mécanismes de communication flexibles et de haute performance pour l'intégration d'un SoC.

4.1.2 Structure du système d'interconnexion

L'évaluation des performances des mécanismes de communication est souvent une des clefs de la réussite dans la conception de SoC efficaces. La recherche montre que les architectures efficaces de communication de haute pèrformance dépendent du domaine d'application visé [47]. Notre recherche démontre comment la gestion des accès d'un arbitre AHB employant plusieurs méthodes d'arbitrage, combiné à un adaptateur générique, peuvent être employés pour l'implémentation d'un bus de haute performance pour l'interconnexion de SoC génériques. Notre but est de prouver qu'AHB convient pour mettre en oeuvre des mécanismes d'interconnexion opérant jusqu'à 0,5 GHz selon le type d'arbitrage, et que cette largeur de bande peut être doublée par la réalisation de plusieurs traitements en parallèle. La solution proposée garantit un accès de haute performance à tous les modules partageant un bus AHB rapide. Elle permet aussi une extension du système d'interconnexion par l'utilisation d'un pont AHB/AHB lié au bus haute performance, comme représenté sur la figure 4.1.

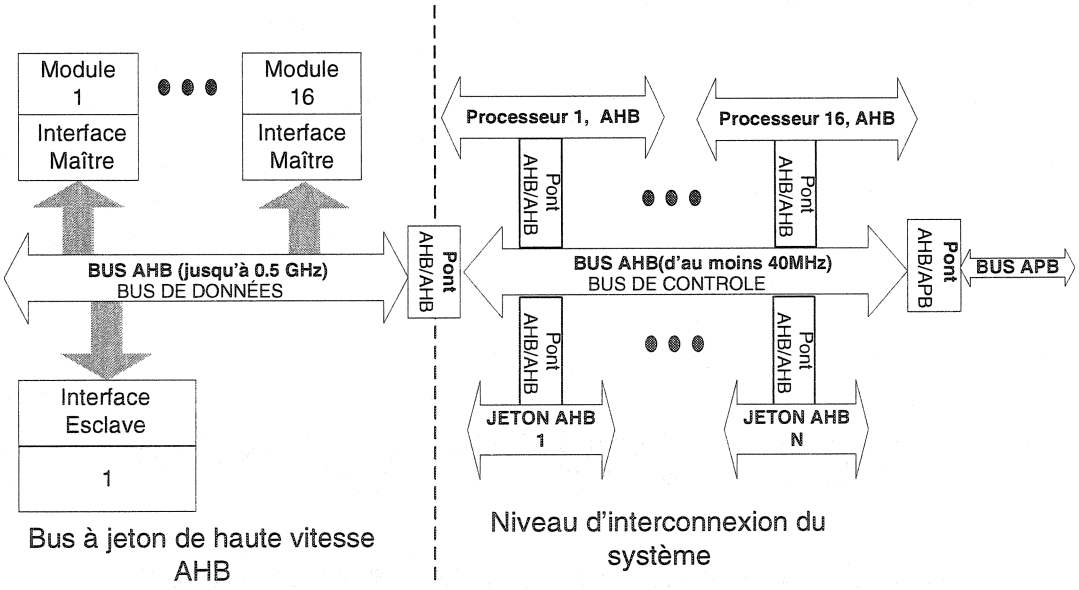


Figure 4.1 Aperçu de la structure proposée

pour une architecture d'interconnexion d'un SoC.

Dans cet exemple, l'architecture est divisée en deux domaines communiquant par l'intermédiaire d'un pont. Le bus de droite est un bus de contrôle. Du côté gauche, le bus est un bus haute performance de communication de haut débit qui peut opérer jusqu'à 0,5 GHz une fois implémentée dans une technologie CMOS 0,18 µm. Notre objectif avec ce bus de données à grande vitesse est d'émuler un système de mémoire comprenant plusieurs ports d'accès de haute performance, en multiplexant des accès à une mémoire de grande vitesse. Le bus permet de partager l'accès à une mémoire embarquée possédant une grande bande passante. Cette mémoire devient alors un mécanisme de communication flexible entre plusieurs modules qui semblent accéder sans latence dans leurs domaines de phase-fréquence respectifs. Ce bus de haute performance permet de supporter des transferts de données apparemment sans latence dans un système comportant jusqu'à seize modules. Cette limite est déterminée par les spécifications du bus AMBA. La largeur de la mémoire est ainsi divisée par le nombre de modules principaux essayant d'y accéder. Le support AHB utilisant des modules qui fonctionnent à différentes fréquences en même temps facilite l'intégration d'un SoC. Pour soutenir ces dispositifs, la bande passante du bus peut être divisée suivant une cédule d'accès appropriée. Le bus peut être utilisé selon une attribution fixe ou partagée pour l'allocation des intervalles de temps qui sont gérés par un arbitre.

4.2 Bus AMBA AHB

Cette section donne un aperçu très général de la norme AMBA-AHB [12] utilisée dans le cadre de notre recherche. La figure suivante présente la structure générale du bus AHB.

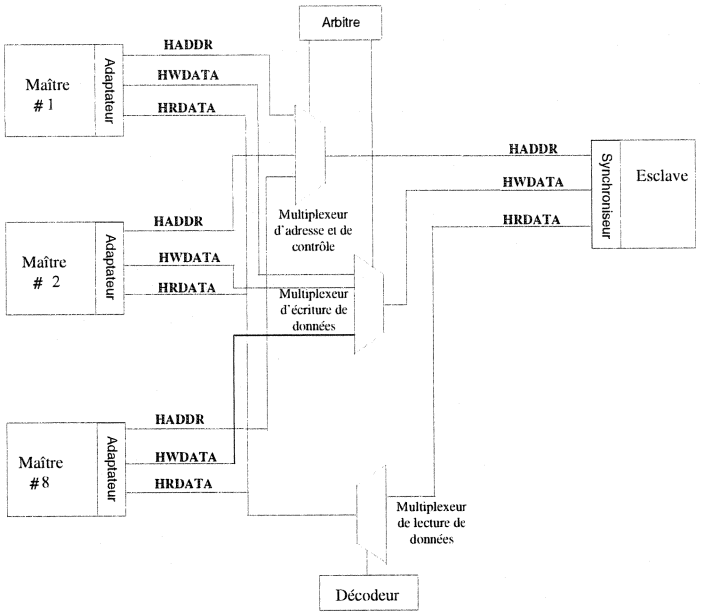


Figure 4.2 Illustration du bus AMBA

Ce schéma montre la séparation en modules maîtres et esclaves ainsi que la présence de décodeurs et d'un arbitre. Il y a trois multiplexeurs pour les adresses, les données en lecture et les données en écriture.

4.2.1 Modules maître et esclave d'un bus AHB

Un seul maître à la fois peut accéder au bus. Ce dernier effectue des opérations de lecture et d'écriture avec des signaux d'adresses, de données et de contrôle. Le module esclave répond aux requêtes d'écriture et de lecture provenant des maîtres. Des méthodes de communication entre les maîtres et les esclaves ont été définies par la norme AMBA.

4.2.2 Décodeur et arbitre

La méthode d'arbitration d'AHB de la norme AMBA est laissée libre au concepteur. L'arbitre du système reçoit les requêtes des modules maîtres et décide par des mécanismes internes qui aura accès au bus. De plus, il s'assure qu'il n'y a qu'un seul maître à la fois qui accède au bus. Le décodeur permet de décoder les adresses provenant des maîtres pour sélectionner un esclave. Nous pouvons obtenir plus de détails sur le sujet en consultant la norme AMBA 2.0 [8].

4.3 Analyse temporelle du signal

4.3.1 Problématique d'interphase et d'optimisation des accès

Le protocole AMBA a une phase d'adresse et une phase de donnée. Pour illustrer la problématique d'interphase (entre les domaines d'horloge), nous allons prendre l'exemple d'une fréquence d'opération de bus 4 fois plus rapide que celle du module. Nous allons examiner l'optimisation requise pour maximiser le nombre de modules ayant accès au bus sans aucune latence. De plus, nous examinerons les cas des accès en lecture et en écriture. Cet exemple est formulé en supposant que les modules qui cherchent à accéder au bus sont régis par des horloges de même phase. La figure 4.3 présente un chronogramme de lecture.

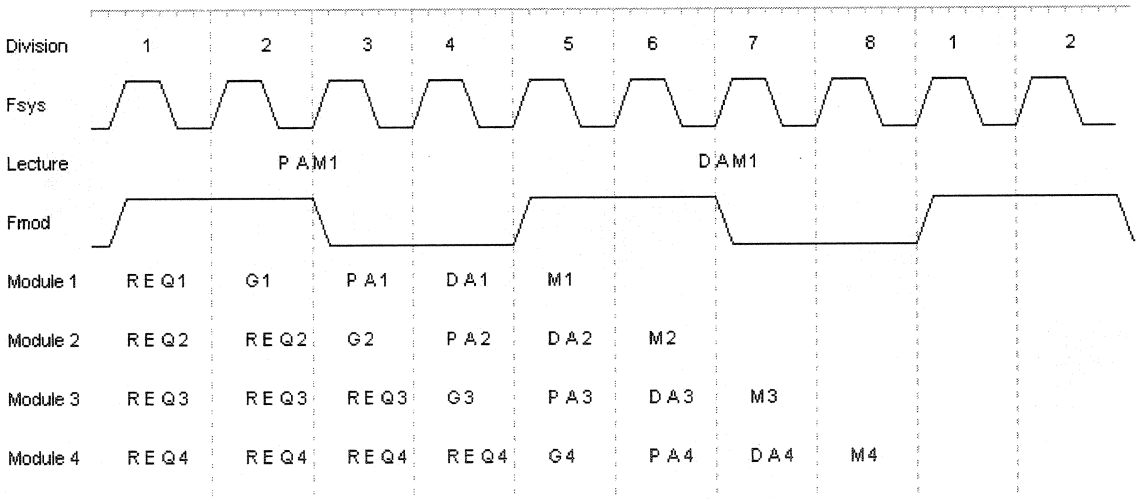


Figure 4.3 Exemple de lecture sur le bus. Fsys est 4 fois plus rapide que Fmod.

Nous avons divisé le temps en huit intervalles (numérotés 1 à 8). Nous définissons cinq états, soit *REQ*, *G*, *PA*, *DA* et *M*. *REQ* correspond à l'émission d'une requête sur le bus. G est un cycle alloué à l'autorisation par l'arbitre d'un accès au bus. *PA* et *DA* sont les phases d'adresse et de données spécifiées par la norme AMBA. M est un cycle alloué pour la mémorisation de chaque valeur provenant du bus. Nous voyons que les modules auront respectivement leurs données dans leur phase de données (*DAM*). Dans l'exemple de la figure 4.3, il y a quatre requêtes simultanées (*REQ1* à *REQ4*) sur le bus dans la

phase d'adresse des modules qui cherchent à accéder au bus. Le dernier module prendra 7 cycles pour recevoir sa donnée (*DA4*). L'utilisation d'AMBA à deux niveaux de pipeline permet d'avoir deux fois plus de temps pour traiter une lecture sur le bus. En mode rafale, la donnée sera mémorisée pour un cycle et il y a aura toujours une cohérence des données lues sur le bus. La lecture ne crée aucun problème d'interphase. Par contre, le problème vient de l'écriture. Cette opération nécessite une hypothèse pour effectuer des requêtes sans latence. La figure 4.4 présente l'opération d'écriture pour les mêmes 4 modules.

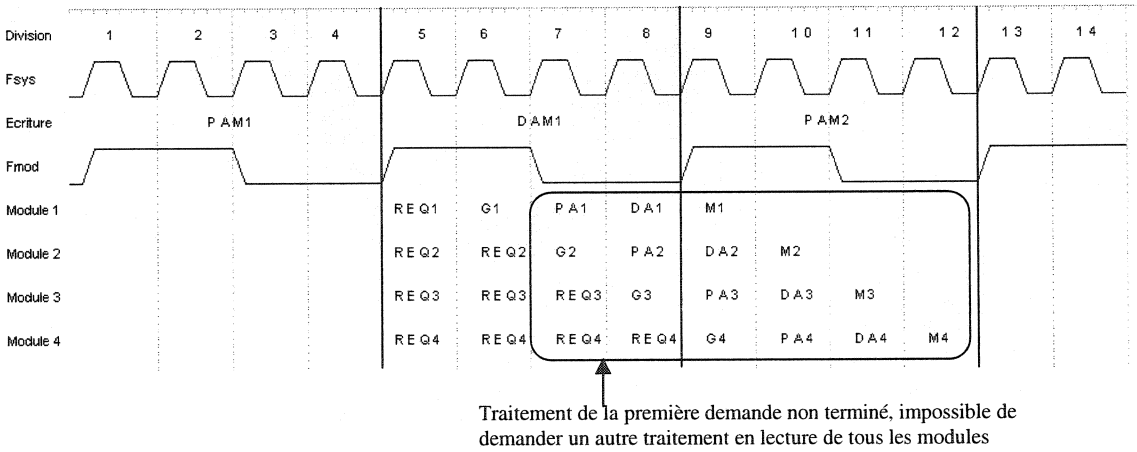


Figure 4.4 Exemple d'écriture sur le bus

Lorsque le module écrit, il envoie l'adresse et ensuite la donnée. Nous ne pouvons pas faire la requête sur le bus immédiatement dans la phase d'adresse (*PAM1*), car il faut attendre la donnée au temps *DAM1*. La demande au bus en écriture sera toujours effectuée dans la phase de donnée, contrairement à la lecture qui sera toujours dans la phase d'adresse. Nous conservons les mêmes états, car il peut y avoir des lectures et des écritures en même temps selon les demandes des modules. Considérons que tous les modules effectuent une requête simultanément au temps *PAM1*, nous aurons toutes les requêtes sur le bus dans l'intervalle de temps numéro cinq. Les quatre premiers cycles ne servent à rien, car il faut attendre que la donnée provenant du module soit arrivée (phase de donnée) avant d'effectuer une transaction d'écriture sur le bus. Nous voulons que quatre modules AMBA en écriture et en lecture communiquent sur un bus à haute vitesse

sans aucune latence. Pour respecter la norme AMBA, il faut que la réponse destinée à chaque module soit valide au plus tard dans l'intervalle de temps numéro neuf ou avant *PAM2*. À ce stade, il est impossible d'avoir quatre modules partageant un multiple de la bande passante du bus. Selon la figure 4-4, seul le premier module a un accès sans latence. Il est nécessaire de faire des hypothèses et d'élaborer des stratégies pour la réalisation d'un bus générique ayant la capacité souhaitée. L'optimisation des latences des modules sera possible grâce à un pont AHB/AHB multi fréquentiel (adaptateur générique). La réalisation de ce dernier permettra d'atteindre nos objectifs. Pour arriver à nos fins, nous supposons qu'il suffit de nous assurer que l'écriture sera faite par le bus avant deux périodes d'horloge du module. Du point de vue de chaque module, l'adaptateur produira les signaux nécessaires pour créer l'illusion qu'une écriture est complétée même si elle ne l'est pas. Ceci permet d'éliminer la latence apparente. Il est possible d'ajouter d'autres mécanismes pour que le module sache réellement s'il y a eu un problème sur le bus. Ceci ne sera pas traité dans cette recherche. Avec cette hypothèse, il y aura une limitation. Nous ne pouvons pas effectuer une opération d'écriture suivie d'une lecture en mode de transfert simple. En effet, l'écriture sur le bus serait alors faite en même temps que la demande de lecture sans latence. Nous verrons plus loin comment il est possible avec la gestion des latences d'éliminer cette contrainte. Nous pouvons voir à la figure 4.3 qu'il existe un maximum au nombre de requêtes au bus que l'on peut servir dans la phase *DAM1* sans créer une latence. C'est-à-dire que si on ajoute un autre module notre réponse appartiendra à l'intervalle de temps 8 +1 et créera une latence. Nous verrons comment on peut utiliser la propriété du pipeline d'AMBA pour doubler le temps disponible pour effectuer les demandes en lecture. Cependant, cette caractéristique n'est d'aucune utilité en écriture.

4.3.2 Zone d'exclusion des modules

Une analyse des problèmes de phases permet de développer une relation générale qui démontre que le système fonctionnera adéquatement. La formule suivante présente les éléments à considérer lors de la réalisation d'un bus générique.

$$Treq + Tgrant + Taddr + Tdata + Tdélai \leq 2\ Tmodule + Latence$$

Tableau 4-1 : Définition des variables qui composent le modèle des contraintes temporelles

Temps	Description
Treq	Le temps nécessaire à l'adaptateur d'obtenir un accès au bus à partir de la requête du module.
Tgrant	Il sera toujours égal à un cycle, c'est l'accès au bus
Taddr et *Tdata*	Les phases d'adresse et de donnée de la norme AMBA
Tdélai	Le délai de la logique combinatoire
Tmodule	La période d'horloge d'un module
Latence	La latence permise d'un module ayant accès au bus.

Notre hypothèse antérieure permet d'avoir une latence (additionnelle) égale à zéro cycle, ce qui correspond du point vue du module à un accès dans un temps toujours inférieur à 2* *Tmodule*. Nous serons dans la zone d'exclusion du module si la formule n'est pas respectée. La fréquence maximale d'un module sera égale à 1/ (*Treq* + *Tgrant* + *Taddr* + *Tdata*) pour ce type de système. La prochaine section présente trois méthodes d'interconnexion des modules soit par accès circulaires, par mémoire et par encodage de priorité.

4.4 Bus par arbitrage circulaire

Notre premier essai effectué en vue de partager les accès à une mémoire haute vitesse exploite un arbitrage circulaire. Les modules communiquent avec le bus par le principe de requêtes et d'accusés de réception.

Les modules mettent leurs adresses et leurs données en phase sur le bus. Dans ce cas, nous avons simplifié les communications entre les modules et le bus. Par contre, le bus communique de la même manière que la norme AMBA en mode de transferts simples. Dans ce premier type de système, l'écriture et la lecture sur le bus s'effectuent dans le même intervalle de temps.

4.4.1 Les synchroniseurs

L'initialisation de la connexion entre ces modules exige une synchronisation délicate afin d'éviter de perdre des données et de créer des goulots d'étranglement. Ainsi, les modules ont été synchronisés selon la norme AMBA. Pour l'adaptateur rapporté ici, nous soutenons seulement le mode de transferts simples (les transactions Split et les états d'attente ne sont pas supportés pour le moment).

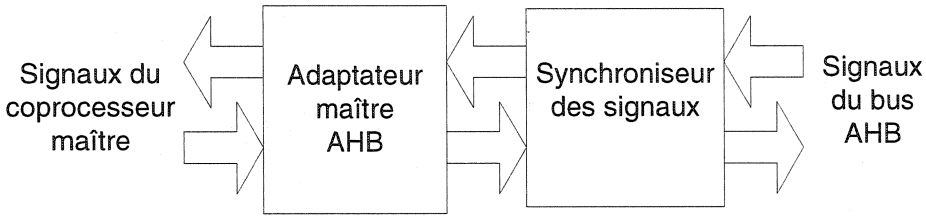


Figure 4.5 Diagramme d'une interface d'un bus AHB

Dans la configuration proposée, le bus rapide et les modules qu'il relie opèrent à des fréquences d'horloge différentes. Pour résoudre ce problème, il est indispensable d'introduire des modules pour la synchronisation des signaux. Le but des synchroniseurs est d'assurer une transmission synchrone entre les signaux provenant des modules à basse fréquence et les signaux du bus AHB, qui opèrent à une fréquence élevée. Le schéma suivant montre la structure de synchronisation.

Il s'agit de transmettre l'adresse *HADDR_X* provenant d'un module maître vers le bus AHB. Le premier multiplexeur et la bascule servent de FIFO. Le signal de demande de transmission *HTRANS_X* provenant d'un module maître sert à sélectionner la valeur *HADDR_X* échantillonnée à la fréquence du bus.

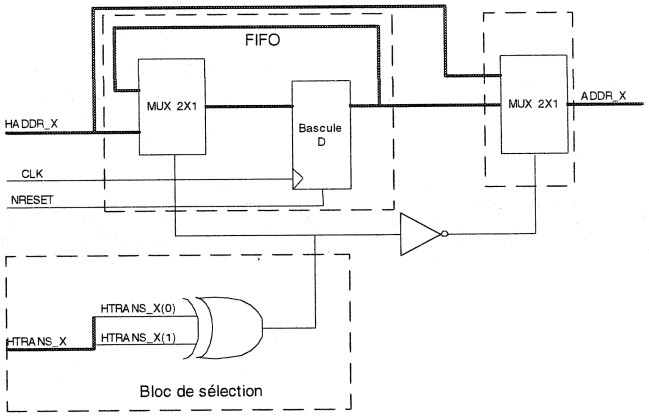


Figure 4.6 Circuit permettant la synchronisation d'adresse

Cette dernière est donc présentée à l'entrée du bus et change d'état au front montant de l'horloge du bus. Tous les synchroniseurs pour les signaux listés au tableau 4-2 ont la même architecture. Ils comportent un FIFO (premier multiplexeur et bascule), un multiplexeur (deuxième multiplexeur) et un bloc de sélection (*XOR*). Cependant, le bloc de sélection diffère selon le type de signal à synchroniser. Le tableau suivant résume les différences entre les blocs de sélection des synchroniseurs.

Tableau 4-2 : Résumé des différents modules de sélection des synchroniseurs

Signal synchronisé	**Description du signal**	**Bloc de sélection**
HADDR_x	Bus d'adresse en provenance du maître x	Porte logique XOR avec les deux bits du signal HTRAN_x
HWDATA_x	Bus de données d'écriture provenant du maître x	Signal HWRITE_x
HRDATA	Bus de données lues provenant de l'esclave (mémoire principale)	HGRANT_x décalé de trois coups d'horloge du bus AHB
HWRITE_x	Signal de contrôle d'écriture provenant du maître x	Porte logique XOR avec les deux bits du signal HTRANS_x
HTRANS_x	Signal de contrôle de type de transfert provenant du maître x	Signal HGRANT_x

Un autre type de synchroniseur est également nécessaire, soit le synchronisateur de réponse illustré à la figure 4-7. Le premier bloc génère une impulsion de synchronisation avec la sélection du maître par l'arbitre. La bascule permet une synchronisation entre les domaines d'horloge. Cela permet de synchroniser la demande avec la fréquence du système.

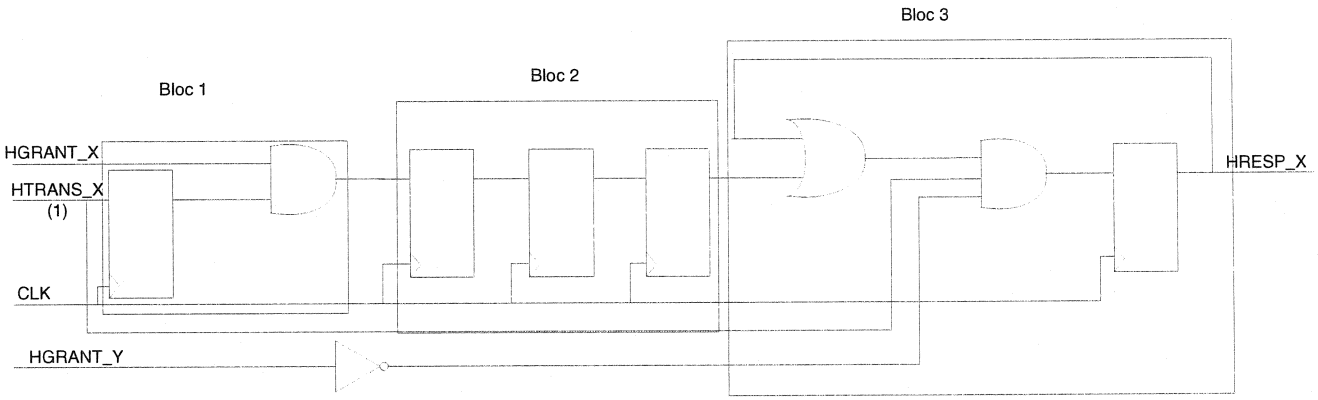


Figure 4.7 Schéma logique du synchronisateur de réponse (*HRESP*)

Le deuxième bloc retarde la réponse. Ce retard est nécessaire pour effectuer la lecture ou l'écriture en mémoire. Il faut 4 cycles du bus rapide au minimum pour générer la réponse destinée au module qui a accès au bus. Ce délai correspond au temps de transfert d'une lecture simple de l'esclave sélectionné vers le maître actif, auquel s'ajoute un cycle de réponse de l'esclave. Le troisième bloc mémorise la réponse et revient à zéro lorsque la requête du maître n'est plus présente. Le système mémorise la réponse pour un cycle d'horloge du module, après quoi elle est annulée. *HGRANT_Y* sert à synchroniser la réponse d'un module à partir d'un accès d'un autre module. Ce signal permet de remettre la réponse à zéro avant d'initier une autre transaction pour permettre de synchroniser notre logique sinon la réponse sera toujours égale à 1.

4.4.2 Arbitrage

Dans ce travail, nous avons implémenté un algorithme d'arbitrage circulaire permettant une rotation de l'accès du bus à chaque cycle. Tous les modules ont alors l'illusion d'accéder à un bus privé fonctionnant à la fréquence des maîtres et esclaves.

L'implémentation de cet algorithme d'arbitrage permet le partage de la ressource mémoire avec une cédule fixe. Ceci est illustré sur le schéma suivant.

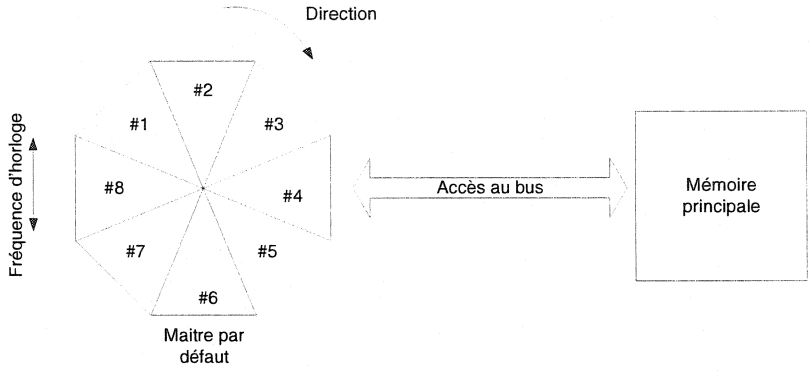


Figure 4.8 Mécanisme d'arbitrage circulaire

Les mécanismes proposés ont été validés par une simulation. Dans cette simulation, nous avons huit modules, dont trois opèrent à 80 MHz et six à 40 MHz. La fréquence du bus au cœur du système est de 320 MHz. À cause d'un arbitrage circulaire et des problèmes d'interphase, il est nécessaire d'intercaler les modules de 40 MHz avec les modules de 80 MHz. Nous voulons cinq modules opérants à 40 MHz avec aucune latence. Nous avons aligné les intervalles de temps pour que la réponse des modules 40 MHz soit toujours dans le même cycle que la réponse. Avec la structure proposée pour le convertisseur de protocole présenté au chapitre précédant, chaque module a l'illusion d'accéder à un port privé d'une mémoire partagée avec plusieurs ports. Ce mécanisme permet de transférer des flux de données à grande vitesse tels que des paquets réseaux ou des trames vidéo. Si la mémoire fournit assez de tampons et que des mécanismes appropriés de commande sont implémentés, il est facile de découpler les modules. Ainsi, chaque module peut traiter les données à son propre rythme, indépendant des autres. Cet arrangement est idéal pour une plate-forme SoC où des modules sont implémentés indépendamment par des concepteurs différents. Les modules opérants à différentes fréquences n'auront pas la même phase. La synchronisation des phases doit être ajustée matériellement de façon manuelle à partir de la simulation.

4.4.3 Résultat de simulation

La figure suivante montre une simulation du bus AHB opérant à 320 MHz. Il montre que les phases d'adresses et de données sont pipelinées. Le schéma suivant présente un résultat d'une simulation où le jeton (dgrant) se décale parmi les huit modules du bus de S1 à S8.

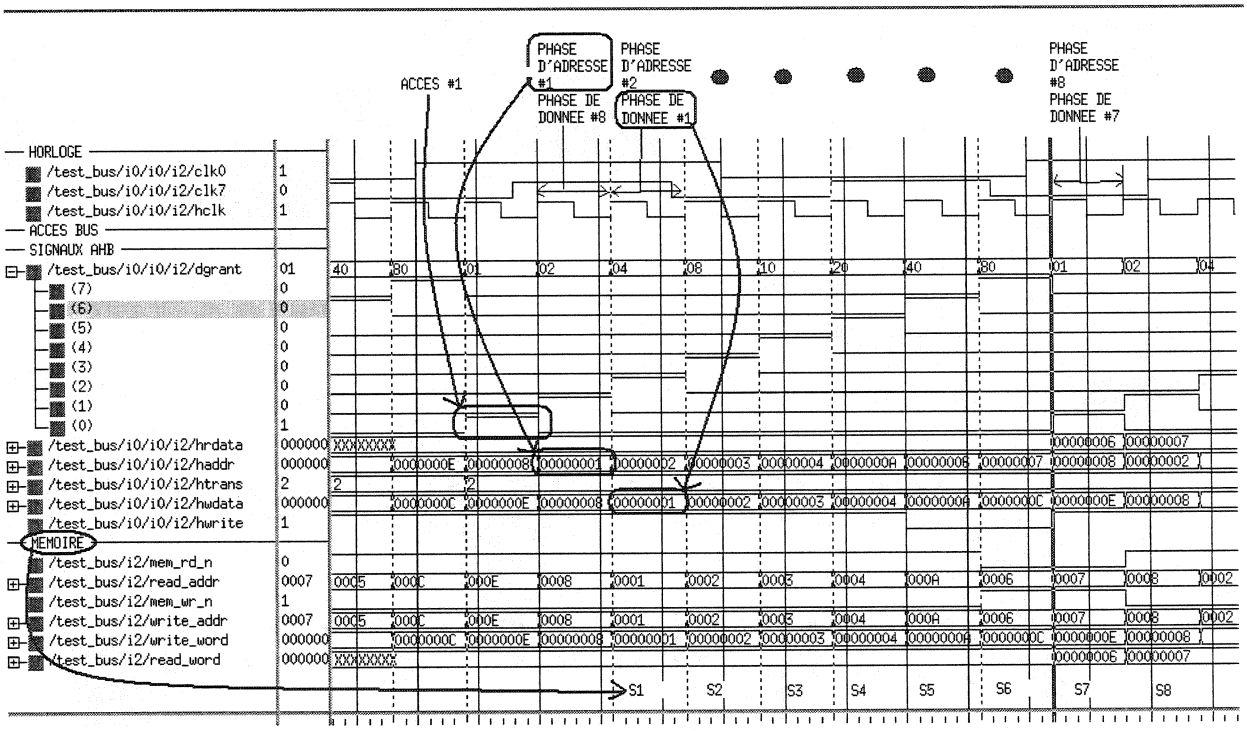


Figure 4.9 Illustration du bus AHB 320 MHz opérant en pipeline

L'arbitre choisit le module qui aura accès au bus sur le prochain cycle d'horloge.

4.4.4 Méthode de vérification

Pour la réalisation de ce type de bus, il est nécessaire d'avoir une méthode de vérification pour valider le concept proposé à partir de la spécification de la communication entre les modules et les mémoires. Nous avons créé des émulateurs maîtres qui permettent de stimuler le comportement d'un module communicant avec la mémoire. La figure suivante présente les résultats du test automatique.

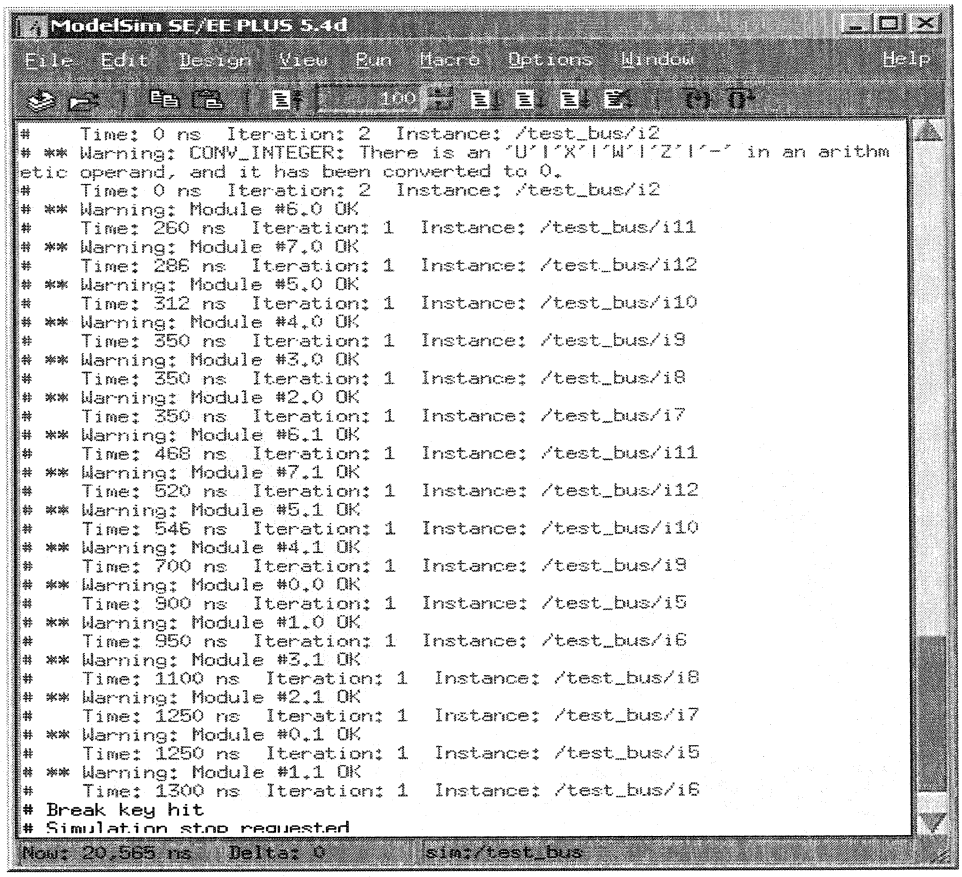


Figure 4.10 Test automatique du bus 320MHz

On s'attend qu'on puisse lire et écrire dans les plages de la mémoire au moyen de notre nouveau bus AMBA géré par arbitrage circulaire. Donc, deux valeurs sont écrites et ensuite lues en mémoire. Nous pouvons voir l'indication 'OK' à coté de chaque transaction pour vérifier si la donnée lue en mémoire correspond à la valeur écrite antérieurement.

4.5 Bus par arbitrage de mémoire

4.5.1 Les synchroniseurs

Nous définirons des nouveaux synchroniseurs qui seront inclus dans un adaptateur générique. Ce dernier sera un pont AMBA AHB/AHB multi fréquentiel avec des fonctionnalités de base. Les synchroniseurs seront les mêmes que ceux présentés dans la prochaine section. La différence de comportement se situe au niveau de l'arbitrage.

Nous utiliserons l'adaptateur générique combiné à un arbitre basé sur la mémoire. Cette technique est développée à titre de comparaison.

4.5.2 Arbitrage

L'arbitrage du bus est réalisé par une mémoire à un port. Cette dernière reçoit les requêtes des modules à son port d'adresse pour donner l'accès à un module à la fois. Ici, on cherche à créer un arbitrage capable de donner des accès aux adaptateurs génériques sans avoir de temps mort. La figure suivante illustre ce propos.

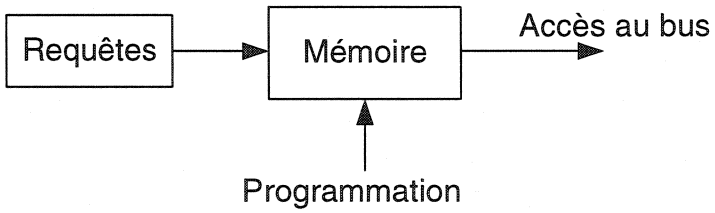


Figure 4.11 Arbitrage par mémoire

Nous pouvons programmer la mémoire selon la bande passante requise. Il est important de mentionner qu'il faut analyser les zones d'exclusion afin d'avoir un système cohérent. Cette méthode nécessite une programmation adéquate. Dans notre cas, nous avons programmé la mémoire de façon à ce qu'elle corresponde à un encodeur de priorité. Dans l'exemple développé, les requêtes arrivent à la même fréquence que le bus peut les servir. Les requêtes sont reçues simultanément des différents modules et chaque requête représente un bit du vecteur d'adresse de la mémoire. Pour faire un encodage de priorité à partir d'une mémoire, il faut mettre dans la case mémoire un bit à un et le reste à zéro. De cette manière, la sortie du bus de données de la mémoire donnera accès à un seul module à la fois suivant un ordre de priorité.

4.5.3 Résultat de simulation

La figure suivante montre les requêtes (*hbusreq*) et les autorisations d'accès mémoire (*hgrant*). Dans les simulations suivantes, nous avons cinq modules dont 3 (0 à 2) opèrent à 80 MHz et 2 (3 et 4) à 40 MHz, ce qui totalise une bande passante de 320 MHz.

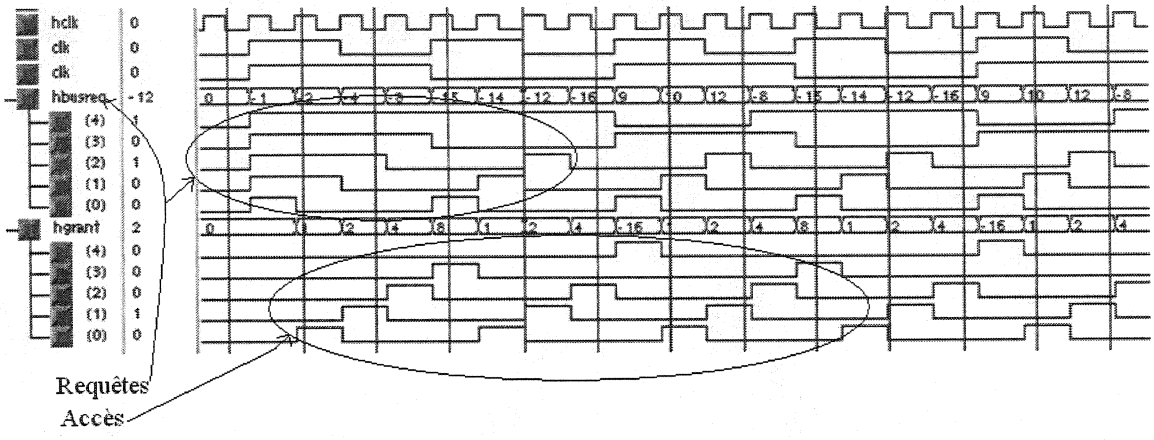


Figure 4.12 Illustration de l'arbitrage par mémoire

Dans ce test, les modules écrivent en rafale. Nous pouvons voir qu'il n'y a aucun cycle d'horloge perdu pour l'attribution des accès. De plus, nous pouvons constater que les modules plus rapides sont traités en premier et que les modules ayant une fréquence inférieure alternent leur accès au bus.

4.5.4 Méthode de vérification

Cette méthode d'arbitrage est considérée à titre de comparaison et elle demande une programmation avec des analyses de phases d'horloge. Nous avons vérifié son fonctionnement manuellement. Une fois la mémoire programmée, nous avons effectué des simulations directement avec Modelsim. Des techniques plus développées seront présentées par la méthode d'encodage de priorité.

4.6 Bus dynamique par encodage de priorité

4.6.1 Les synchroniseurs de l'adaptateur générique

La flexibilité de communication entre des modules opérant à différentes fréquences nécessite un adaptateur générique capable de communiquer simultanément dans des domaines d'horloge différents. La réalisation d'un pont AMBA multi fréquentiel et auto synchronisant permet d'augmenter la capacité du bus à haut débit développé précédemment. L'objectif est de créer une interface AMBA générique de haute

performance acceptant diverses fréquences combinées avec un arbitrage adéquat. AMBA est un protocole à deux niveaux de pipeline. Il est nécessaire de connaître l'état de la requête et à quel niveau de pipeline elle appartient pour éviter de mélanger les données et d'être dans une zone d'horloge incohérente.

4.6.1.1 Les Machines à états

Pour la synchronisation entre les phases d'horloge de différentes fréquences, nous utiliserons un registre à décalage (RD) décalant un 1 logique opérant à la fréquence du module permettant de connaître la phase. Ce dernier est le cœur de cette synchronisation. De plus, il faut deux machines à états permettant la synchronisation entre les modules et le bus à haute vitesse. La figure suivante présente la première machine à états (FSM) pour une communication avec le bus. Cette machine opère à la même fréquence que le bus.

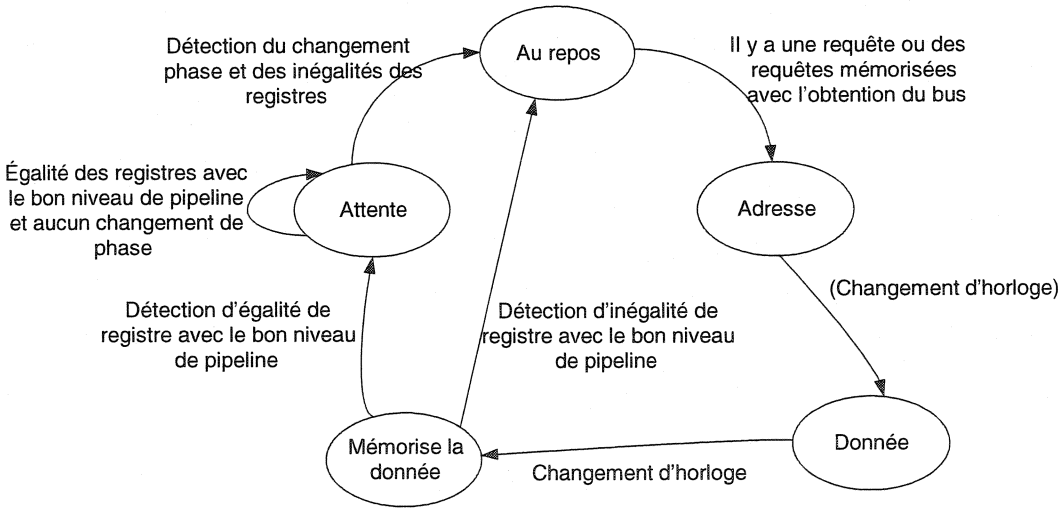


Figure 4.13 Machine à états du bus dans l'adaptateur

Nous retrouvons les états du protocole AMBA, soit la phase d'adresse et la phase de donnée. De plus, il y a des états internes au module permettant de mémoriser les données provenant du bus. L'état attente permet d'assurer une synchronisation adéquate. Le fonctionnement de cette machine à états sera expliqué plus en détail ultérieurement. La machine à états suivante permet de communiquer avec le module. Cette machine opère à la même fréquence que le module.

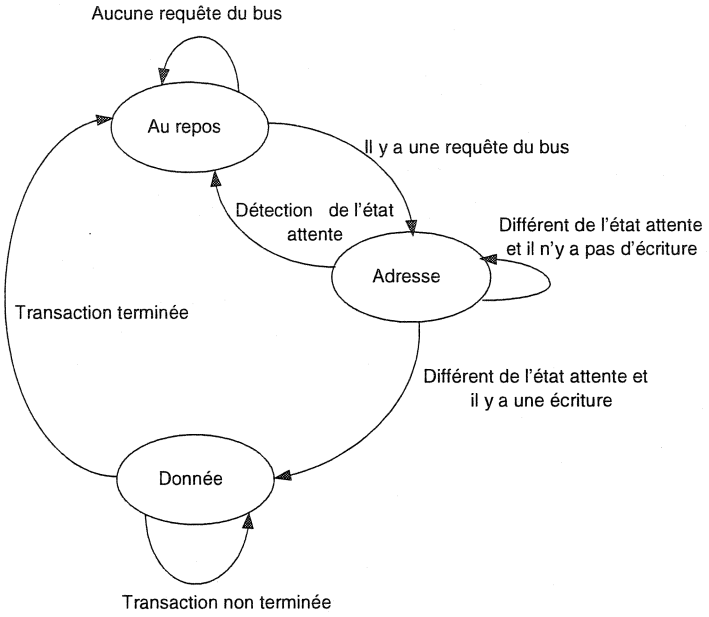


Figure 4.14 Machine à état du module dans l'adaptateur

Nous trouvons les phases d'adresse et de données d'AMBA. Lors de la phase d'adresse, il y a une détection pour savoir s'il y a une lecture ou une écriture. La requête du bus sera faite pour une lecture dans la phase d'adresse et pour une écriture dans la phase de donnée.

4.6.1.2 Les signaux de contrôle

Pour la gestion de l'adaptateur, nous avons besoin de deux signaux de contrôle. Le premier s'inversera lors du passage à l'état d'adresse de la machine à états finis du bus de la figure 4.16 et l'autre lors de chaque détection d'une nouvelle requête. Les circuits des figures 4.15 et 4.16 permettent la génération des signaux de contrôle requis.

Cette détection est primordiale pour savoir à quel niveau de pipeline la requête va appartenir. Le mécanisme inverse le signal à chaque fois que le multiplexeur est activé pour un cycle. Il y aura inversion du signal lorsqu'il y a détection de l'événement choisi. Il faut mémoriser l'état du registre à décalage pour connaître les phases adéquates pour la

synchronisation des données. De cette façon, nous pouvons connaître la phase relative du module par rapport à celle du bus de haute vitesse.

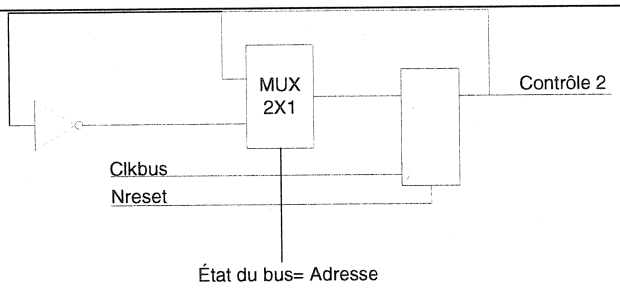


Figure 4.15 Circuit de contrôle 2 de l'adaptateur

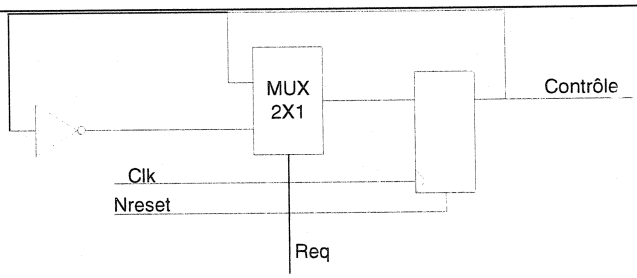


Figure 4.16 Circuit de contrôle de l'adaptateur

Le circuit suivant permet de mémoriser l'état du registre RD de référence dans des mémoires temporaires. Il faut mentionner qu'il y a une circuiterie identique pour chacune des mémoires, soit *Mémoire1* et *Mémoire2*.

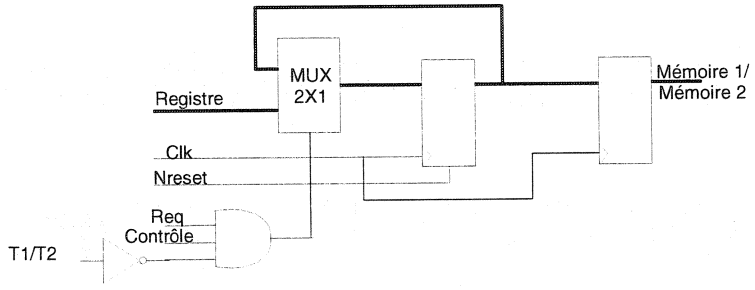


Figure 4.17 Temporisation du registre

Lors d'une requête, le contenu registre à décalage sera mis dans une des deux mémoires pour conserver l'information de la phase. À ce moment, l'autre mémoire peut contenir l'information provenant d'une requête antérieure dont le traitement ne serait pas encore complétée. Il y a deux mémoires puisque nous avons deux niveaux de pipeline avec AMBA. À chaque requête, nous commuterons les mémoires pour la temporisation du registre RD. C'est donc dire que la première requête va dans la mémoire 1 et la deuxième dans la mémoire 2 et ainsi de suite. Le schéma de la figure 4.18 présente la méthode de génération des signaux *Ctrl1* et *Ctrl2* permettant une communication des registres.

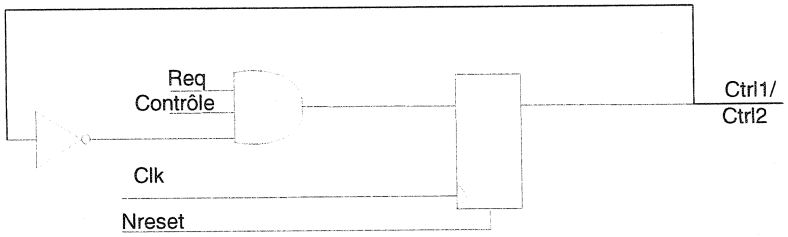


Figure 4.18 Génération des signaux *Ctrl1* et *Ctrl2*

Une autre manière d'expliquer le fonctionnement général est la suivante. Le signal contrôle commutera à chaque détection d'une requête vers le bus du système. Ces signaux sont importants, car ils permettent de savoir si une transaction est complétée ou non et ils permettent la mémorisation de RD dans les mémoires. Ainsi, les mémoires conservent l'information sur la phase du module si une transaction n'est pas terminée, sinon ils peuvent mémoriser les informations d'une autre requête. Cette dernière est nécessaire pour un fonctionnement adéquat en mode transfert en rafale. Par la suite, les mémoires (mémoire1 et mémoire2) seront synchronisées adéquatement par la détection de la requête suivante en utilisant les signaux *Ctrl1* et *Ctrl2*. L'illustration suivante permet de montrer le fonctionnement de ce principe.

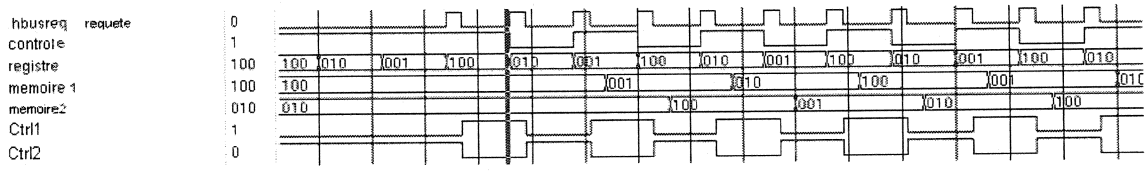


Figure 4.19 Illustration de la commutation du bus

Nous pouvons voir que le vecteur registre change à tous les cycles et que les vecteurs *mémoire1* et *mémoire2* alternent leur mémorisation du registre à décalage. Cette méthode permet de connaître l'état de la demande par rapport à la requête du module.

4.6.1.3 Les mécanismes d'écriture sur le bus

Une fois les signaux de contrôle interne créés pour l'adaptateur générique, nous avons besoin de mécanismes de lecture et d'écriture sur le bus. La figure 4-20 présente un circuit permettant de mettre en phase la donnée à écrire sur le bus par rapport à son

module. Il faut écrire l'adresse et la donnée sur le bus. Le contrôle s'effectuera avec les mémoires et le signal *contrôle2*. *Contrôle2* permet la détection d'une variation des requêtes. C'est-à-dire qu'il commutera à chaque détection de l'état adresse du bus. À partir des signaux de contrôle et de la valeur de la phase du module par rapport au bus, la donnée sera mise sur le bus ou sera retardé d'un cycle.

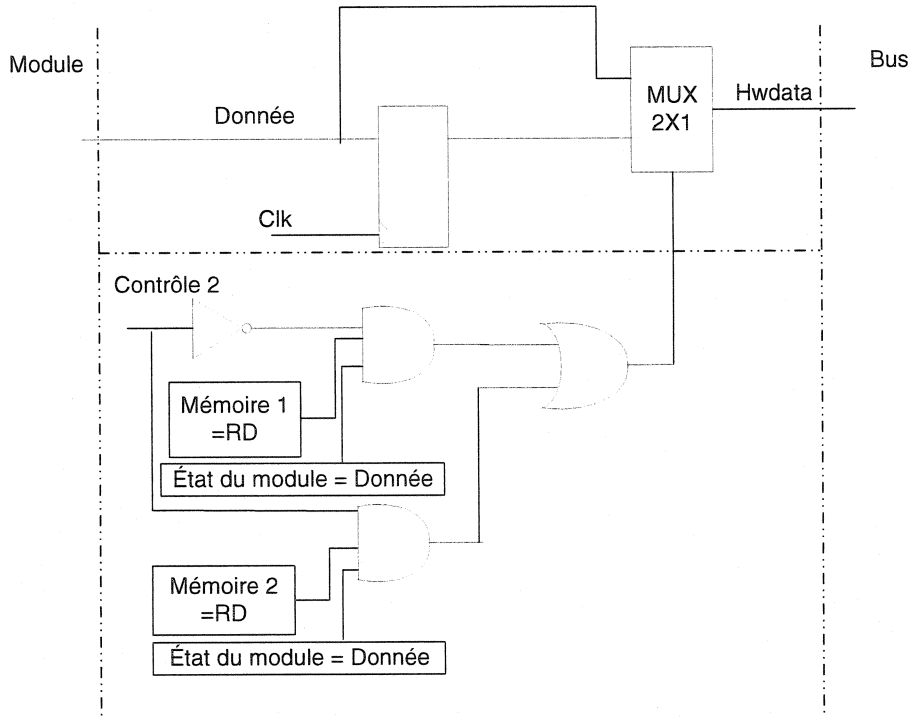


Figure 4.20 Synchronisateur de l'écriture de *Hwdata*

Il y aura changement de la valeur à écrire lorsque la FSM du module sera égale à la phase de donnée. Nous pouvons constater qu'il y a une double logique attachée à contrôle2, à cause des deux niveaux de pipeline d'AMBA. Il est donc nécessaire d'avoir un autre synchroniseur capable de gérer le signal d'adresse.

La figure 4-21 présente la synchronisation de l'adresse permettant la lecture et l'écriture sur le bus. Nous retrouvons le même mécanisme de contrôle d'un multiplexeur. Nous avons un module communicant avec l'adaptateur générique selon la norme AMBA et un bus de haute vitesse AMBA. Il est nécessaire de mettre les phases d'adresse et de

données ensemble avant d'effectuer une requête sur le bus. La première bascule (*D1*) sert à placer la phase d'adresse et la phase de données ensemble.

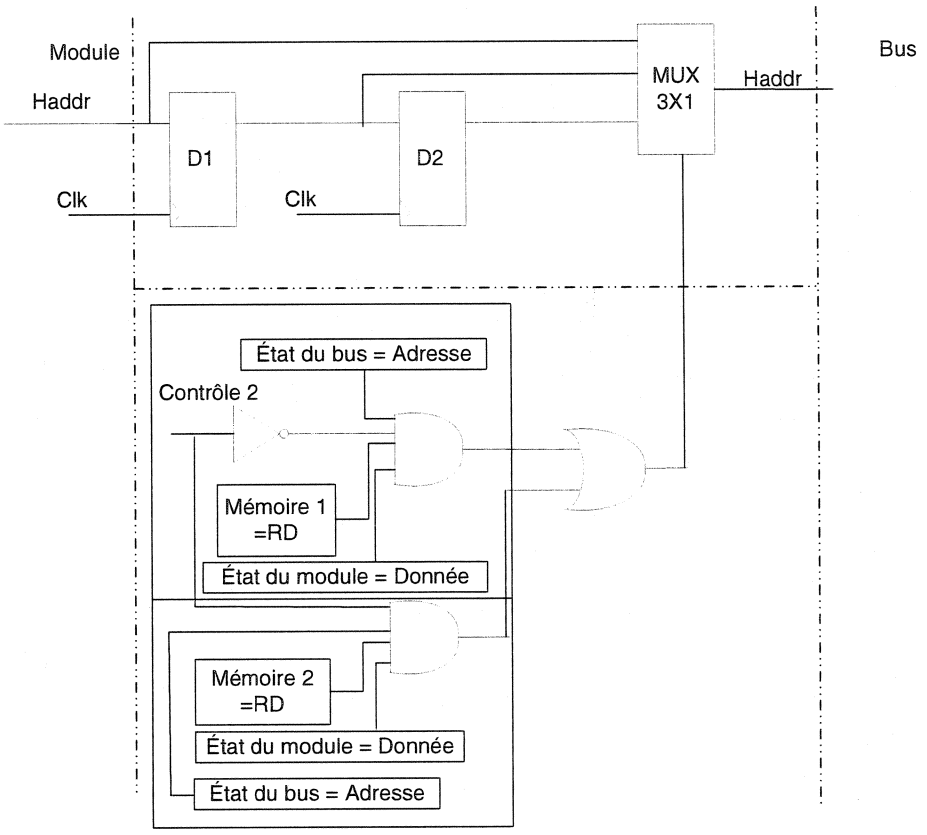


Figure 4.21 Synchroniseur d'adresse

La deuxième bascule (*D2*) sert à conserver la valeur si le bus n'a pas eu le temps d'effectuer la requête. Dans ce cas ci, l'adaptateur dira au module que la requête a été bien réalisée, mais il le fera plus tard. Le module pourra réaliser une autre requête même si celle en cours n'est pas terminée. Lors de l'analyse temporelle discutée antérieurement, nous avons vu qu'il faut effectuer une anticipation de la réponse pour éviter une latence. Dans notre application, nous voulons un transfert en rafale en écriture ou en lecture. L'écriture se fera sur le bus dans un temps approprié, en garantissant toujours la cohérence des données pour l'écriture. Lors d'une lecture, *D1* et *D2* ne seront pas nécessaires, le multiplexeur passera l'adresse provenant du module directement sur le bus. Un dernier synchronisateur important est celui de la lecture de la donnée.

4.6.1.4 Les mécanismes de lecture sur le bus

La figure 4-22 présente le circuit de synchronisation de la lecture.

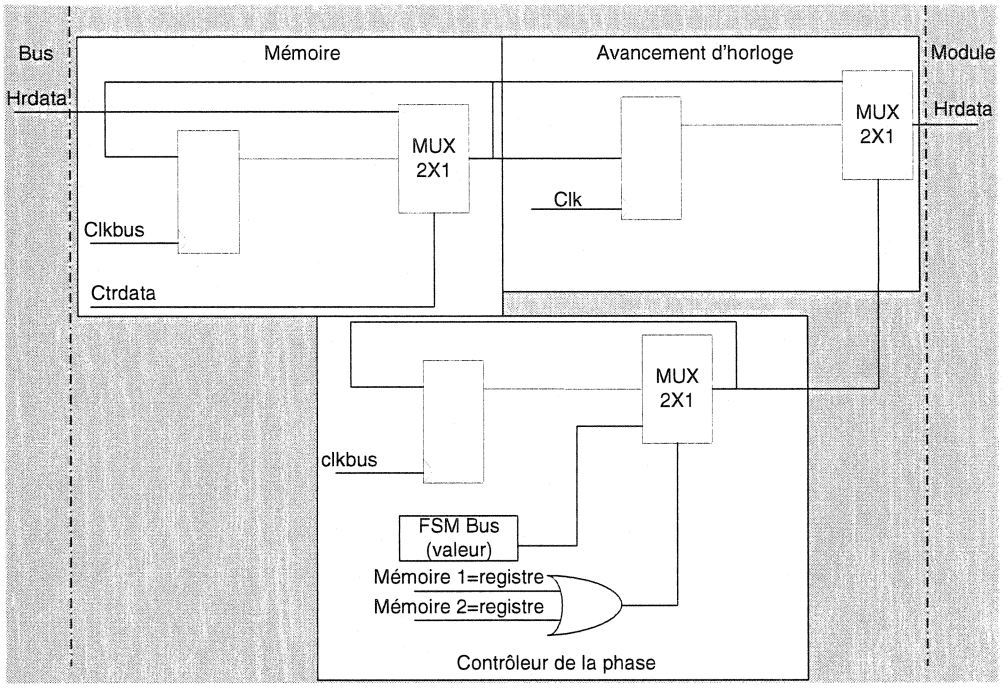


Figure 4.22 Synchroniseur de lecture

Nous voulons récupérer la valeur lue sur le bus. Il y a trois mécanismes pour ce synchroniseur, soit un contrôleur de la phase, un pour l'avancement d'horloge et une mémoire. Le contrôleur de phase gère l'avancement de la donnée au module. L'avancement d'horloge permet de décaler la donnée d'un cycle. En terminant, une mémoire est requise pour temporiser la valeur lue sur le bus de haute vitesse et la conserver jusqu'à ce que le module puisse la lire. Une fois que notre adaptateur est prêt, il faut un arbitre pour permettre une communication adéquate sur le bus.

4.6.2 Arbitrage

Le choix de l'arbitrage du bus AMBA est important pour le bon fonctionnement du système. L'illustration de la figure 4.23 permet de voir la disposition des modules par rapport au mécanisme d'arbitrage choisi.

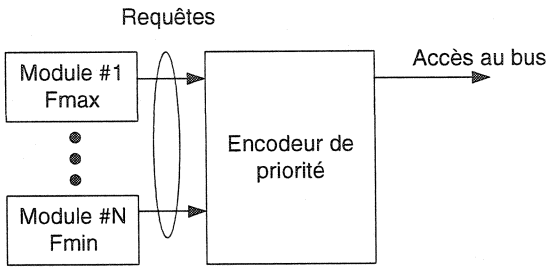


Figure 4.23 Arbitrage par encodeur de priorité

À partir des deux versions précédentes, nous sommes arrivés à notre modèle proposé, soit l'encodage de priorité. La prochaine section permettra d'avoir une vue d'ensemble des possibilités de cette nouvelle approche de conception d'un bus AMBA.

4.6.3 Résultats de simulation

Les résultats de simulation permettent de voir les possibilités d'une combinaison entre l'adaptateur générique et un arbitrage dynamique. Nous verrons trois principaux résultats. Le premier confirme la possibilité de combiner des modules opérant à des fréquences différentes. Le deuxième confirme la vitesse d'opération du bus en relation avec l'ensemble des modules desservis. Le troisième confirme la fonctionnalité de l'arbitre. Les diverses possibilités offertes par le bus sont listées au tableau 4-3.

Tableau 4-3 : Possibilités de combinaison de modules offertes par le bus implémenté

Exemples	Modules	Fréquence des modules
1	8	40 MHz
2	6	53 MHz
3	4	80 MHz
4	3	80 MHz
	2	40 MHz
5	2	80 MHz
	4	40 MHz

Dans nos simulations, nous avons un bus opérant à 320MHz et nous en distribuons la bande passante de différentes manières aux modules qui composent le système. Il n'y a pas de latence, car la somme des fréquences des modules ne dépasse pas la fréquence du bus et le bus peut répondre immédiatement à toutes les requêtes. Ce bus supporte de façon flexible et efficace un environnement de communication standard. Plusieurs modules compatibles avec AMBA et opérant à différentes fréquences selon leurs spécifications peuvent s'échanger des données sans nécessiter de tampon de communication entre le bus et le module. Le matériel nécessaire à la réalisation du pont AHB/AHB est invariable en fonction de la fréquence. C'est-à-dire que la surface requise par l'adaptateur est fixe. La figure 4-24 présente la capacité de la fréquence maximale du bus en fonction du nombre de modules connectés.

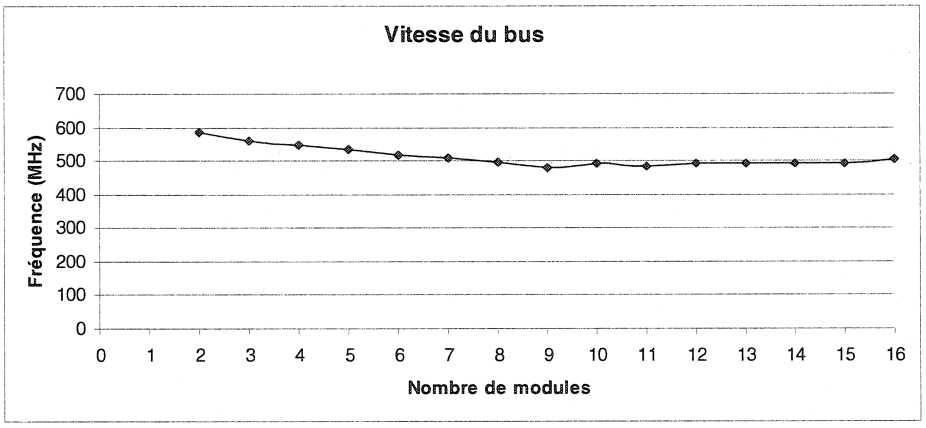


Figure 4.24 Fréquence du bus en fonction des modules

Nous pouvons constater qu'il y a une diminution de la fréquence d'opération avec le nombre de modules quand il y a moins de neuf modules. Cependant, la fréquence reste autour de 500 MHz lorsqu'il y a plus de neuf modules. Les résultats ont été obtenus avec l'outil de Design Analyser. Plus on augmente le nombre de modules, plus il y a de la logique combinatoire pour le multiplexage et le contrôle. La synthèse du modèle est faite de façon générique par l'outil de synthèse, et les fréquences représentent des estimations. Le graphique précédant permet de décider selon l'application et la plate-forme voulue la répartition du nombre de modules et du nombre de mémoires. Par exemple, nous pouvons prendre huit modules partageant une bande passante de 500 MHz avec une seule

mémoire. Pour une même fonctionnalité orthogonale, nous pouvons multiplier par deux la bande passante avec deux mémoires. Dans ce second cas, nous aurons deux segments de quatre modules partageant une bande passante de 525 MHz. Ce système aurait une bande passante de 1050 MHz. Nous pouvons mettre un module commun aux deux mémoires pour permettre l'échange entre tous les modules. Les résultats au niveau de l'arbitrage sont les mêmes que ceux pour la mémoire et nous obtenons les mêmes chronogrammes. Pour augmenter la capacité fréquentielle du bus dans la suite de cette recherche, il est important de connaître ses limitations. D'après les analyses post synthèse, le chemin critique est sur le signal d'adresse. C'est-à-dire que nous avons comparé les fichiers résultants de la synthèse du bus. La limitation de la fréquence est toujours déterminée par le chemin entre le registre *mémoire2* et le bus d'adresse. Les détails concernant ce chemin sont fournis à l'Annexe B.

4.6.4 Méthode de vérification

Les méthodes de validation (attente vs spécification) et de vérification (spécification vs implémentation) sont très importantes pour le bus générique multi fréquentiel. Nous avons changé la stratégie de simulation, car nous voulons une méthode automatique pour vérifier son fonctionnement. La description de l'arbitre qui fonctionne par encodage de priorité ne nécessite aucune modification du code lorsqu'on change les fréquences des modules. Le premier changement apporté par rapport à la méthode de vérification du bus circulaire est de créer des vecteurs d'entrée et de sortie en VHDL. Ceci permet d'interfacer un module directement au bus sans être obligé de connecter manuellement les fils. À partir de la norme AMBA, nous avons pris son package VHDL pour l'intégrer dans notre modèle de simulation. Deuxièmement, il faut réaliser des générateurs de test AMBA attachés à des registres de données. La figure suivante montre la topologie proposée pour cette vérification.

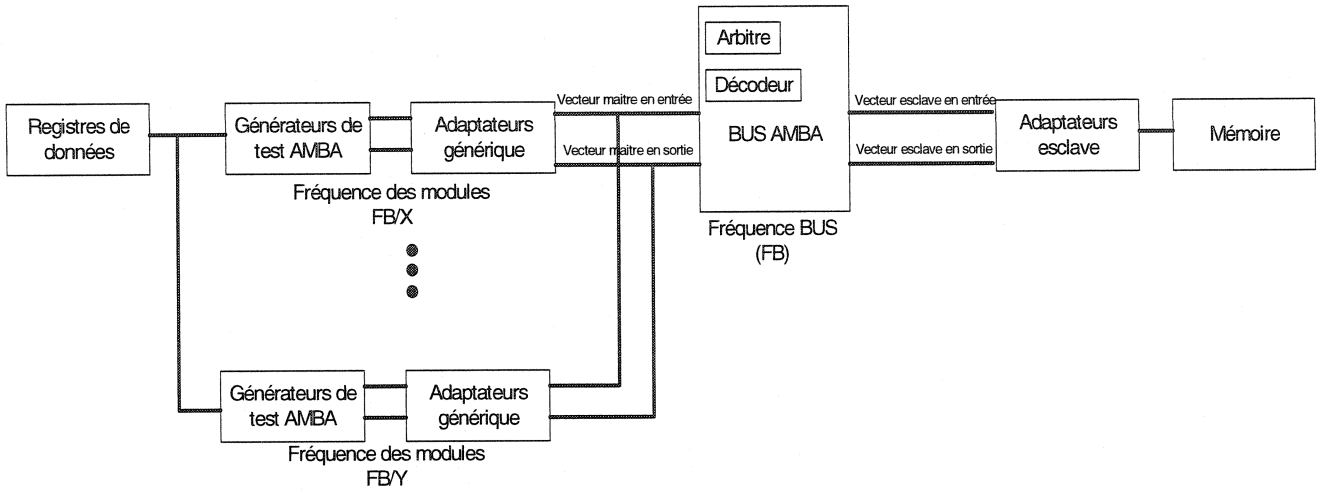


Figure 4.25 Schéma du système

Le registre de données fournit aux générateurs de test AMBA les valeurs initiales d'adresses et de données. Les générateurs sont des modules maîtres AMBA et ils génèrent les stimuli à partir de l'information des registres de données. Ils testent la lecture avec l'écriture en mode simple et en mode rafale. Il y a aura réplication automatique du module de génération de test AMBA et de l'adaptateur générique. Un domaine d'horloge représente une fréquence d'opération de plusieurs modules. Sur la figure 4-25, nous pouvons voir la fréquence du bus (*FB*) et il y a deux domaines d'horloge définis par *FB/X* et *FB/Y*. *X* et *Y* sont des multiples de *FB*. Par exemple, un système opérant à 320MHz avec des modules opérants à une fréquence de 80MHz et de 40MHz auront respectivement *X* et *Y* égaux à 4 et 8. Il est important de mettre le domaine d'horloge le plus rapide en premier et ensuite le moins vite. Pour réaliser une vérification automatique, nous changerons la valeur de *X* et de *Y* ainsi que le nombre de modules par domaine d'horloge à l'aide d'un programme. Le cheminement pour vérification avant et après synthèse est illustré à la figure 4-26.

Il y a un script simulant le bus avant la synthèse permettant de changer les paramètres tel que les horloges et le nombre de modules par domaine d'horloge. Il y a aussi un autre fichier principal pour la simulation après synthèse. Il faut vérifier les rapports générés par l'outil de cadence pour qu'il n'y ait aucun problème.

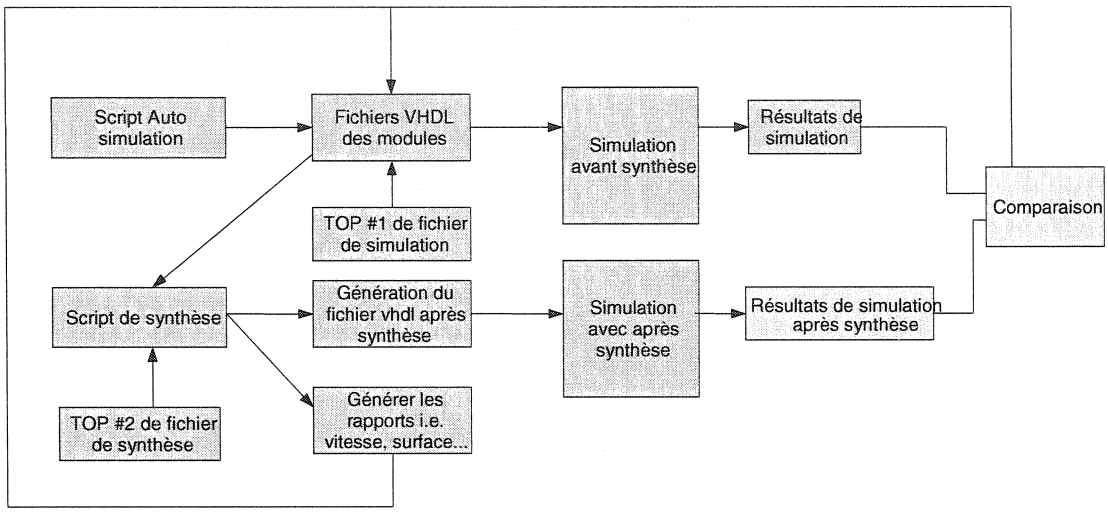


Figure 4.26 Schéma de vérification du bus

Le script synthétise le VHDL avec les paramètres requis et simule ensuite son comportement. La vérification effectuée compare les résultats avant et après synthèse. Il faut valider que les résultats avant et après synthèse sont identiques. De plus, nous obtenons les informations concernant la vitesse, la surface, etc. La méthode proposée et implémentée vérifie automatique le système pour chaque ensemble de paramètres que l'on désire simuler.

4.7 Comparaison des trois méthodes

Les méthodes basées sur la mémoire et l'encodeur de priorité requièrent l'adaptateur générique pour fonctionner. L'arbitrage circulaire a ses propres synchroniseurs qui ne sont pas flexibles. Le tableau suivant présente une comparaison des mécanismes d'accès au bus. Le mode circulaire est fixe et requiert un ajustement matériel propre au module. Il faut relier correctement le signal *HGRANT_Y* au module correspondant. Ce dernier ne requiert pas que le module effectue une requête contrairement aux deux autres. Le mécanisme d'accès géré par une mémoire nécessite une programmation initiale. D'après nos résultats, le mode circulaire opère à une fréquence plus élevée que les autres méthodes. La limitation fréquentielle du bus pour la mémoire et l'encodeur de priorité est la même, soit le chemin d'adressage. À partir des résultats de l'arbitrage circulaire, nous avons créé un système d'interconnexion flexible et modulaire. C'est-à-dire que

l'interconnexion d'un bus AHB reste identique et que chaque module possède en apparence un bus privé AMBA correspondant à sa fréquence d'opération.

Tableau 4-4 : Comparaison des mécanismes d'interconnexion

Arbitres	Programmable	Modules sans latence	Demande une analyse
Circulaire	Non	Oui (limité)	Oui
Mémoire	Oui	Oui	Oui
Encodeur de priorité	Non	Oui	Non

Le pont AHB/AHB appelé l'adaptateur générique entre le bus et le module permet une plus grande flexibilité de conception. Nous avons changé le type d'arbitrage de la mémoire à l'encodeur de priorité sans changer les adaptateurs. L'encodeur de priorité combiné avec l'adaptateur permet d'avoir un système auto synchronisant. La mémoire permet plus de souplesse dynamique que l'encodeur. La meilleure solution jusqu'à maintenant est l'encodeur de priorité ne nécessitant aucune analyse et pas de programmation. La bande passante des trois méthodes est fixe. Dans le cas de l'encodeur de priorité, il est possible avec une gestion des latences d'avoir un ensemble de modules dont le potentiel total de demande d'accès au bus requiert une bande passante plus grande que la capacité du bus.

4.8 Conclusion

Nous avons exploré les manières possibles d'augmenter la capacité d'un bus de haute performance basé sur le protocole AMBA. La clef du succès dans un système sur puce est d'avoir une interconnexion entre les modules flexibles et de haute performance (hauts débits et basses latences). Pour cette réalisation, nous avons utilisé une mémoire de haute performance permettant d'opérer à grande vitesse. Les modules partagent la bande passante de cette dernière. Nous pouvons voir en annexe B qu'il est possible d'obtenir une vitesse de plus de 500 MHz après le placement et le routage avec une arbitration circulaire offrant peu de flexibilité. La création d'un pont AHB/AHB générique multi-

fréquentiel nous a permis d'avoir un système auto synchronisant et modulaire, dont la fréquence d'opération a été estimée à 500MHz sur la base d'analyses de délais statiques réalisées sur le circuit avant placement et routage. Le bus permet d'échanger sur demande des données à haut débit entre les modules du système. Les concepteurs peuvent développer leurs modules de façon concurrente. Dans le futur, nous pouvons attacher à l'adaptateur deux autres modules, le premier est un anticipateur de donnée permettant d'avoir des modules opérant à de plus grandes fréquences sur le bus. Le deuxième est un gestionnaire de réponse permettant une latence dans les temps de réponse. De plus, un encodeur de priorité avec un algorithme basé sur le code thermomètre permettrait d'avoir une bande passante fixe et dynamique simultanément sur le bus.

CHAPITRE 5
CONCLUSION ET TRAVAUX FUTURS

5.1 Conclusion sur les travaux

Dans cette recherche, nous avons étudié différents langages de conception et les méthodes d'interconnexion des modules dans un système sur puce. Une modélisation hétérogène à double profilage a été proposée ainsi qu'une plate-forme de référence pour le développement des applications souhaitées. Nous avons étudié les caractéristiques communes entre les langages pour les connecter ensemble. À partir de ces informations, nous avons créé une méthode de conception partant de la théorie jusqu'à son implémentation physique. De plus, nous avons réalisé avec la méthode une plate-forme de traitement vidéo en exploitant l'algorithme de Wiener comme cas d'étude. De même, nous avons exploré les niveaux d'abstraction et les langages qui se prêtent le mieux à un ensemble de tâches de conception. Dans la méthode proposée, il y a deux profilages. Le premier profilage se fait au niveau algorithmique et le deuxième au niveau du processeur. Les contraintes du système et sa performance peuvent être analysées par l'outil Seamless. Les informations recueillies peuvent être l'utilisation du bus, les résultats de profilage du code, l'utilisation de la mémoire etc. Une fois la méthode appliquée, il reste le logiciel opérant sur un processeur et le matériel. Nous avons présenté une co-vérification matérielle et logicielle de notre plate-forme de référence. La méthode proposée permet une simulation hétérogène et le raffinement des modules uniquement, car les interconnexions entre les modules sont déjà réalisées par un bus générique de haute performance. Nous avons défini une plate-forme de référence possédant au minimum deux bus. Le premier bus sert au contrôle de la plate-forme et il opère à une basse vitesse alors que le deuxième permet le traitement à haute vitesse de l'application souhaitée. Les deux bus sont reliés par un pont. Il peut aussi y avoir un pont reliant un bus de basse fréquence acceptant des périphériques lents.

Le raffinement des communications du système a été décrit au quatrième chapitre. Pour ce faire, nous avons utilisé un exemple concret d'interconnexion d'un SoC ciblant une

plate-forme pour la conversion de protocoles. Nous avons créé un bus capable, à l'aide d'adaptateurs génériques, une forme de pont AHB/AHB multi-fréquentiel, de connecter à une mémoire principale des modules opérant à différentes fréquences de façon transparente au concepteur. Nous avons exploré trois méthodes d'interconnexion soit l'allocation circulaire, l'allocation par mémoire et l'allocation par encodage de priorité. Nous avons discuté de leurs caractéristiques et de leur fonctionnement. Cette recherche a permis de prouver qu'il est possible d'avoir un bus générique et opérant à haute vitesse. De plus, les interconnexions des modules sont largement compatibles avec la norme industrielle AMBA. Dans notre cas, AMBA a été adopté à cause de sa simplicité de fonctionnement. Cette recherche a permis d'identifier les limitations de la vitesse d'opération et la proposition d'un système d'interconnexion auto synchronisant au module et permettant d'avoir une mémoire d'un seul port. La nouvelle méthode de conception avec un bus de haute performance permet la réutilisation de composants déjà développés et une réduction du temps de développement. Dans cette recherche, deux plates-formes ont été développées. Ces plates-formes ciblent deux applications distinctes. La première est un convertisseur de protocoles et le deuxième est un VPM (Video processing module). Les deux se basent sur la plate-forme de référence et utilisent les méthodes d'interconnexion développées dans cette recherche. La structure multi-fréquentielle du bus supporte une architecture générique qui a été validée par des simulations. La structure garantit la bande passante à chaque module qui échange des données. Elle permet d'implémenter chaque module en isolation des autres. Chaque module peut donc être développé par des concepteurs différents et cela de façon concurrente. Les principales contributions de cette recherche sont l'analyse, la conception, la réalisation de divers types de bus avec arbitration comportant un adaptateur multi-fréquentiel. De plus, nous avons élaboré une méthode à double profilage qui supporte la communication entre divers langages. Cette méthode offre une flexibilité accrue pour l'analyse et l'optimisation d'architecture hétérogène. Les infrastrutures logicielles (ARM7) et matérielles de la plate-forme de transmission et de la plate-forme de traitement vidéo constituent aussi des contributions de ce mémoire.

5.2 Limitations et recherche future

Les éléments essentiels de base ont été réalisés. Cependant, il existe une limitation en fréquence d'opération du bus AHB à cause de son chemin d'adresse. De plus, un module ne peut effectuer une écriture suivie d'une lecture sans latence à cause de sa caractéristique. Le pont multi-fréquentiel AHB/AHB est une version de base ne gérant pas la latence. Dans notre application, nous n'avons pas eu besoin de la traiter. De plus, l'implémentation du pont permet un accès simple ou en rafale. Nous avons implémenté quelques modes en rafale. Une application peut requérir d'autres modes que l'on pourrait dériver en ajoutant des caractéristiques dans la machine à états finis appropriée. Dans cette recherche, nous avons élaboré théoriquement d'autres possibilités d'arbitrage qui n'ont pas été implémentées, tel que l'arbitrage par encodage thermomètre par rétroaction ou non. Ce dernier permet d'avoir une bande passante dynamique et statique simultanément. Le terme dynamique veut dire que le mode en rafale peut être redirigé temporairement vers un module dans le cas ou la somme des bandes passantes demandées est supérieure à celle du bus. De plus, la fréquence d'opération pourrait être augmentée en brisant le chemin d'adresse et en utilisant une anticipation des adresses. Une autre possibilité future est la connexion de modules opérant à des fréquences plus élevées que celles permises dans ce mémoire. Nous pourrions le faire en permettant une latence d'un cycle au mode en rafale. Cette recherche pourrait conduire à une structure générique et modulaire permettant d'ajouter d'autres modules tel que le gestionnaire de latence et l'anticipateur de données. Nous avons créé une méthode et une structure de base permettant la création d'autres plates-formes dans un temps de conception plus court.

RÉFÉRENCES

[1] ALPHAMOSAIC; "Nucleus operating system"; http://www.alphamosaic.com/products/software/os/nucleus.php

[2] ALPHAMOSAIC; "Usage Scenarios: Videophone and PDA"; http://www.alphamosaic.com/applications/videophone.pdf

[3] ALPHAMOSAIC; "Videocore the versatile video processor"; http://www.alphamosaic.com/products/appnotes/casestudy.pdf

[4] ALPHEY J.; BAXTER C.; COONELL J.; GOODENOUGH J.; HARRIS A.; LENNARD C.; MATHEWSON B.; NIGHTINGALE A.; THORNTON I.; TOPPING K.; "Star-IP Centric Platforms for SOC"; Winning the SoC Revolution by G. Martin and H. Chang, Kluwer Academic Plublishers, 2003, Chap. 5.

[5] AUGUSTO DE OLIVEIRA J.; VAN ANTWERPEN H.; "The Philips Nexperia digital video platform"; Winning the SoC Revolution by G. Martin and H. Chang, Kluwer Academic Plublishers, 2003, Chap. 4.

[6] ANDREWS M.; "Creating a C-Bridge Pin Interface Model" , Mentor Graphics Technical Library

[7] AUGUIN M. ; " Introduction à la Conception de SoC"; Roscoff Université de Nice Sophia Antipolis - CNRS, Ecole Thématique 2003.

[8] ARM ; "AMBA Specification''; Technical specification, Doc N0: ARMIHI-0011A, Issued : May 2001.

[9] ARM; "PrimeXsysPlatforms"; http://www.arm.com/products/solutions/PrimeXsysPlatforms.html

[10] BAILEY B.; "Co-Verification: From Tool to Methodology"; DesignCon January 2002.

[11] BERG S.; "Using Seamless C-Bridge in the SystemC Simulation Environment", Mentor Graphics Technical Library.

[12] BERTOLA M.; "Conception, réalisation et étude d'une plate-forme générique basée sur le protocole AMBA AHB"; École polytechnique, Avril 2003.

[13] BOIS G., FILION L., TSIKHANOVICH A., ABOULHAMID E .M., Modélisation, raffinement et techniques de programmation orientée objet avec SystemC, La spécification et la validation des systèmes hétérogènes embarqués, A.A. JERRAYA et G. NICOLESCU (ed.), Hermes, fall 2003.

[14] BOMBANA M.; BRUSHI F.; "SystemC-VHDL co-simulation and synthesis in the HW domain"; Proceedings of the Design, Automation and Test in Europe Conference and Exhibition IEEE 2003.

[15] CERNY E.; "Introduction to SystemVerilog 3.1"; Ecole Polytechnique-ReSMiQ, avec la collaboration de IEEE Montréal - Solid-State Circuits Chapter, 17 juin 2003.

[16] DIGUET J.P.; GOGNIAT G.; MARTIN E. ; "Codesign ou Conception Conjointe Logiciel/Matériel"; LESTER - Université de Bretagne Sud.

[17] FUJITSU LIMITED; FUJITSUN LABORATORIES LIMITED; "Develops New SoC Design Methodology Based on UML and C Programming Languages"; Tokyo, April 16, 2002.

[18] GROTKER T.; "Transaction Level Modeling with SystemC"; www-ti.informatik.uni-tuebingen.de/~systemc/Documents/Presentation-7-TLM_groetker.pdf.

[19] GROTKER T.; LIAO S.; MARTIN G.; SWAN S.; "System Design with SystemC''; Kluwer Academic Publishers, Boston,Hardbound, 2002.

[20] IBM; CoreConnect Bus Architecture; http://www.chips.ibm.com/products/ coreconnect.

[21] IBM; "PowerNP NPe405 Embedded Processors PowerPC based network processor"; Technical Papers, IBM, 2003.

[22] KAYE R.; "Seamless with C-bridge: "C" Based Co-Verification"; Technical Papers, Mentor, p.27, 17 September 2002.

[23] LEEF S.; "A Methodology for Virtual Hardware/Software Integration"; Technical Papers, Mentor Graphics Corporation.

[24] LIM J.S.; "Two-Dimensional Signal and Image Processing"; Prentice Hall, 1990.

[25] NICOLESCU N.; GABRIELA E.; "Spécification et validation des systèmes hétérogènes embarqués Thèse de doctorat TIMA, Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs"; Institut national polytechnique de grenoble - inpg, Novembre 2002.

[26] MATLAB; http://www.mathworks.com

[27] MARSHALL D.; "Programming in C UNIX System Calls and Subroutines using C", http://www.cs.cf.ac.uk/Dave/C

[28] MARTIN G.; "UML for Embedded Systems Specification and Design: Motivation and Overview"; ACM SIGDA, Paris, France, March 4-8, 2002

[29] MARTIN G.; "SystemC Tools"; European systemC users group meeting, Lago Maggiore 2002.

[30] MENTOR GRAPHICS CORPORATION; "Seamless CVE User's and Reference Manual Software Version 5.0 Document Version 5.0_0.1"; 2003.

[31] MENTOR GRAPHIC; "HDL designer Series"; Datasheet, 2003.

[32] MODELSIM; "Modelsim User's manuel version 5.7f"; 03juin 2003.

[33] OPEN CORE; www.opencores.org.

[34] OPEN SYSTEMC INITIATIVE (OSCI); "SystemC version 2.0 documentation"; http://www.systemc.org, 2003.

[35] PHILIPS; "Home entertainment engine for advanced analog/digital and digital TV, Nexperia PNX8550"; Littérature de Philips; Décembre 2003.

[36] PHILIPS; "Home Entertainment Engine Nexperia pnx8525"; Littérature de Philips; http://www.semiconductors.philips.com/acrobat/literature/9397/75008276. pdf.; 2001.

[37] PRYOR D.; "The Role of C-Based Languages in SoC Flows" , Mentor Graphics Technical Library, 2002.

[38] RABAEY J.M.; SANGIO-VINCENTELLI A.; "System-on-a-Chip - A Platform Perspective"; Proceedings 9th Korean Conference on Semiconductors, April 2002.

[39] SAVARIA,Y.; "Conception et vérification des circuits VLSI''; Éditions de l'École Polytechnique de Montréal, 1988.

[40] SYSTEMVERILOG; http:///www.systemverilog.com

[41] SYSTEMC; http:///www.systemc.org

[42] VEAL R. L.; PETROSIAN L.; STOLLON N. S.; "Multi-core SoC Platform Integration using AMBA"; Proceedings of DesignCon2002 System-on-Chip and IP Design Conference, January 2002.

[43] VSIA; http://vsia.org

[44] XANALYS INCORPORATED; "LispWorks Foreign Language Interface User Guide and Reference Manual Version 4.2"; December 2001.

[45] XILINX; "Digital Visual Interface Technology, Implementations, and Applications"; www.xilinx.com/esp/dvt/cdv/collateral/dvi_dvt.pdf.

[46] XI C.; NINGYI X.; ZUCHENG Z.; "A Methodology for SystemC Algorithmic Model Verification Applying MATLAB" IEEE 2003.

[47] ZHANG L., CHAUDHARY V.; "On the performance of Bus Interconnection for SOCs"; Workshop on Media and Stream Processors, Istanbul, Turkey; November 2002.

ANNEXE A

Modélisation hétérogène

Matlab

Adaptateur #1	Adaptateur #2

int matlab_add(int tmp)
{
//Création des matrices Matlab
mxArray
*OUT=NULL,*IN_MX=NULL;
//Formatage des données
double IN[1];
IN[0]=tmp;
//Initialisation du module
InitializeModule_add();
//Création d'une matrice Matlab
IN_MX =
mxCreateDoubleMatrix(1,1,mxREAL);
//Copie les données dans la matrice
memcpy(mxGetPr(IN_MX),
IN,sizeof(double));
//Réalisation de la fonction
OUT = mlfAdd(IN_MX);
 	
//Copie du résultat
memcpy(IN,mxGetPr(OUT),
sizeof(double));
tmp=IN[0];
/* Destruction des matrices*/
mxDestroyArray(OUT);
mxDestroyArray(IN_MX);
/* Destruction du module*/
TerminateModule_add();
return(tmp);
}
	int enginedemo(int tmp,Engine *ep)
	{
	//Création des matrices Matlab
	mxArray *IN = NULL, *OUT = NULL;
	//Formatage des données
	double data[1] ={10};
	data[0] = tmp;
	
	//Création d'une matrice Matlab
	IN = mxCreateDoubleMatrix(1, 1, mxREAL);
	//Copie les données dans la matrice
	memcpy(mxGetPr(IN),data, sizeof(double));
	//Réalisation de la fonction
	engPutVariable(ep, "IN", IN);
	engEvalString(ep, "OUT=mrank(IN)");
	OUT=engGetVariable(ep, "OUT");
	//Copie du résultat
	memcpy(data,mxGetPr(OUT), sizeof(double));
	tmp=data[0];
	/* Destruction des matrices*/
	mxDestroyArray(IN);
	mxDestroyArray(OUT);
	
	return tmp;
	}

Le tableau suivant présente les codes de chacun.

SystemC (Serveur)	FLI Modelsim (Client)

//Iniatialisation
Fonction : Début du programme principal
port_sk = tcp_passive_open(port);
if (port_sk < 0)
{ perror("socket"); exit(1); }
/*Attente d'une connexion d'un client*/
client_sk = tcp_accept(port_sk);
/* Accepte seulement un client, donc fermeture du port */
close(port_sk);
 	
Fonction : for(;;;)
len=read(client_sk,&syn,sizeof(structcom));
if (len==0){break; }
//Mettre le code à réaliser
write(client_sk,&syn,sizeof(struct com));
	//initialisation du TCI/IP
	Dans la fonction : c_input_module_init
	serv_sk = tcp_active_open(host,port);
	if (serv_sk < 0)
	{ perror("socket"); exit(1); }
	
	Fonction : proc_clock_CB
	write(serv_sk,&syn,sizeof(struct com));
	/* wait for serverÕs message */
	len=read(serv_sk,&syn,sizeof(struct com));

Mémoire partagé SystemC et VHDL

SystemC (Serveur)	FLI Modelsim (Client)

//Phase d'initialisation
//Attribution de la clef
key = 5678;
//Création de la plage.
if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
{ perror("shmget"); exit(1); }
 	
//Nous attachons la plage mémoire à notre environnement
shm = (struct com *) shmat (shmid, (struct com *) 0, 0);
 	
//Nous mettons quelque chose dans la mémoire
s = shm;
//Déclaration des composantes
sc_initialize();
//Indique la présence du simulateur
write();
//Indique la présence du simulateur
read();
printf("Lecture %d \n",syn.synch);
 	
printf("DEMARRAGE\n");
 	
for(;;)
{
read();
//metre le code a réalisé
 	
write();
 	
}
	//Phase d'initialisation
	//Attribution de la clef
	key = 5678;
	//Création de la plage.
	if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
	perror("shmget");
	exit(1);
	}
	
	// Nous attachons la plage mémoire a notre environnement
	if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
	{perror("shmat"); exit(1); }
	
	//attente du server
	while(syn.synch!=40)
	{
	syn=*shm;
	sleep(1);
	//printf("toto");
	}
	//Réaliser un acquitter de réception
	syn.synch=2;
	//Écriture en mémoire
	*shm=syn;
	
	Fonction : proc_clock_CB
	// Écriture, 2 sert à la synchronisation
	syn.synch=2;
	*shm=syn;
	while(syn.synch!=40)
	{
	syn=*shm;
	}

Extrait d'une communication SystemC avec le PIM

```
/* Define a function to perform the model functionality */

void ExecuteMyModel(void *modelData, viaUint32 numcycles)
        {
        /* Code*/
        if (reset_val == 0) {

       /* Code à initialiser*/

      else  {

//TCP/IP
write(serv_sk,&syn,sizeof(struct com));
 /*  wait for serverÕs message  */
len = read(serv_sk,&syn,sizeof(struct com));

/* Code synchrone*/
```

```
//-------------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------------
--------
// File          : main_sc.cpp
// Author        : Mathieu Dubois
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois. This model is the confidential
and
// proprietary property of Mathieu Dubois and the possession or use of
this
// file requires a written license from Mathieu Dubois.
//-------------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------------
---------


#include <systemc.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <bloc.h>
#include <mean.h>
#include <localmean.h>
#include <localvar.h>
#include <filter.h>
#include <image.h>

// Fonction principale
int sc_main( int argc, char* argv[] )
{


//Declaration des fifo
  sc_fifo<bloc> fimage;
  sc_fifo<bloc> fimage2;
  sc_fifo<bloc> fimage3;
  sc_fifo<bloc> flm(10);
  sc_fifo<bloc> flm2(10);
  sc_fifo<bloc> flv(10);
  sc_fifo<bloc> flv2(10);
  sc_fifo<bloc> imfiltree(10);
  sc_fifo<double> fnoise(10);
```

```
//Declaration des modules
localmean _localmean("_localmean");
localvar _localvar("_localvar");
mean _mean("_mean");
filter _filter("_filter");
image _image("image");

_localmean.in(fimage);
_localmean.out(flm);
_localmean.out2(flm2);
_localvar.in(fimage2);
_localvar.in2(flm2);
_localvar.out(flv);
_localvar.out2(flv2);
_filter.plm(flm);
_filter.pim(fimage3);
_filter.pnoise(fnoise);
_filter.plv(flv);
_filter.out(imfiltree);
_mean.in(flv2);
_mean.out(fnoise);
_image.out(fimage);
_image.out2(fimage2);
_image.out3(fimage3);
_image.in(imfiltree);

        sc_start(100000);


      printf("fin de la simulation");

      return 0;
}
```

```
//-------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------
--------
// File          : bloc.cpp
// Author        : Mathieu Dubois
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois. This model is the confidential
and
// proprietary property of Mathieu Dubois and the possession or use of
this
// file requires a written license from Mathieu Dubois.
//-------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------
---------

#include <systemc.h>

#ifndef BLOC_H
#define BLOC_H

class bloc
{
public:
      // CTOR
      bloc();
      bloc(const bloc&);
      bloc& operator=(const bloc&);
      virtual ~bloc();

 // Surchage de l'opérateur de sortie à l'écran
  friend ostream& operator << (ostream & o, const bloc & p);

  void set_image_height (unsigned int);
  void set_image_width (unsigned int);
  void set_my_image(int *);
  void set_nhood(unsigned int);

   double**      my_image;

 /// Filtering neighborhood (3x3 / 5x5 / 7x7)
  unsigned int nhood;
```

```
  /// Image to be filtered

  /// Image height (nb of row)
  unsigned int image_height;
  /// Image width (nb of col)
  unsigned int image_width;


};

#endif
```

```
//-------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------
--------
// File          : mean.cpp
// Author        : Mathieu Dubois & Serge Catudal
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois & Serge Catudal. This model is
the confidential and
// proprietary property of Mathieu Dubois & Serge Catudal and the
possession or use of this
// file requires a written license from Mathieu Dubois & Serge Catudal.
//-------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------
---------

#include "mean.h"

void mean::traitement()
{
      while(true)
      {
               in->read(donnee);
                 cout<<"mean"<<donnee<<endl;
// result = sum of all matrix elements / number of matrix elements
    for(row=0; row<donnee.image_height; row++)
    for(col=0; col<donnee.image_width; col++)
    result = result + donnee.my_image[row][col];
    result = result / (donnee.image_height * donnee.image_width);
         cout<<"mean_fin"<<result<<endl;
            out->write(result);

             wait(1,SC_NS);


      }
}
```

```
//-------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------
--------
// File          : localmean.cpp
// Author        : Mathieu Dubois & Serge Catudal
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois & Serge Catudal. This model is
the confidential and
// proprietary property of Mathieu Dubois & Serge Catudal and the
possession or use of this
// file requires a written license from Mathieu Dubois & Serge Catudal.
//-------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------
---------

#include "math.h"
#include "localmean.h"

void localmean::traitement()
{
      while(true)
      {
              in->read(donnee);
                cout<<"localmean"<<donnee;
                donnee=filter2D(1,donnee);
                cout<<"localmean_fin"<<donnee;
                out->write(donnee);
                out2->write(donnee);

                wait(1,SC_NS);


      }
}

bloc localmean::filter2D(unsigned int p,bloc pdonnee)
{

 unsigned int row;
  unsigned int col;
  int          ext;
```

```
  int**         m_filter;
  double**      result;
  double**      g;
  int           l,r,u,d;
  int           m,n;


cout<<"filter_in"<<pdonnee;
  // Initialisation du filtre 2D
  ext = (pdonnee.nhood - 1) / 2;
  m_filter = new int*[pdonnee.nhood];
  for(row=0; row<pdonnee.nhood; row++){
    m_filter[row] = new int[pdonnee.nhood];
    for(col=0; col<pdonnee.nhood; col++)
      m_filter[row][col] = 1;
  }

  // Initialisation de la matrice g et result
  g      = new double*[pdonnee.image_height];
  result = new double*[pdonnee.image_height];
  for(row=0; row<pdonnee.image_height; row++){
    g[row]      = new double[pdonnee.image_width];
    result[row] = new double[pdonnee.image_width];
    for(col=0; col<pdonnee.image_width; col++)
      g[row][col] = pow(pdonnee.my_image[row][col], p);
  }

  // Filtrage de la matrice temporaire
  for(row=0; row<pdonnee.image_height; row++)
    for(col=0; col<pdonnee.image_width; col++){
      // Determination des bornes utilisables de la matrice de filtrage
      if(row == 0){
        u = ext; d = pdonnee.nhood;
      } else if(row == 1 && (pdonnee.nhood == 5 || pdonnee.nhood ==
7)){
        u = ext - 1; d = pdonnee.nhood;
      } else if(row == 2 && pdonnee.nhood == 7){
        u = ext - 2; d =pdonnee.nhood;
      } else if(row == (pdonnee.image_height - 3) && pdonnee.nhood ==
7){
        u = 0; d = pdonnee.nhood - ext + 2;
  }else if(row == (pdonnee.image_height - 2) && (pdonnee.nhood == 5 ||
pdonnee.nhood ==7)){
        u = 0; d = pdonnee.nhood - ext + 1;
      } else if(row == (pdonnee.image_height - 1)){
        u = 0; d = pdonnee.nhood - ext;
      } else {
        u = 0; d = pdonnee.nhood;
      }

     if(col == 0){
        l = ext; r = pdonnee.nhood;
      } else if(col == 1 && (pdonnee.nhood == 5 || pdonnee.nhood ==
7)){
```

```
        l = ext - 1; r = pdonnee.nhood;
      } else if(col == 2 && pdonnee.nhood == 7){
        l = ext - 2; r = pdonnee.nhood;
      } else if(col == (pdonnee.image_width - 3) && pdonnee.nhood ==
7){
        l = 0; r = pdonnee.nhood - ext + 2;
  } else if(col == (pdonnee.image_width - 2) && (pdonnee.nhood == 5 ||
pdonnee.nhood ==7)){
        l = 0; r = pdonnee.nhood - ext + 1;
      } else if(col == (pdonnee.image_width - 1)){
        l = 0; r = pdonnee.nhood - ext;
      } else {
        l = 0; r = pdonnee.nhood;
      }

     // Apply 2D filter
     result[row][col] = 0;
     for(m=u-ext; m<d-ext; m++)
       for(n=l-ext; n<r-ext; n++)
         result[row][col] = result[row][col] + (g[row+m][col+n] *
m_filter[m+ext][n+ext]);
     result[row][col] = result[row][col] / (pdonnee.nhood *
pdonnee.nhood);
    }

cout<<"nhood"<<pdonnee.nhood<<endl;
cout<<"resultat"<<result[20][20]<<endl;
pdonnee.my_image=result;
  return pdonnee;




}
```

```
//-------------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------------
--------
// File          : local.h
// Author        : Mathieu Dubois
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois. This model is the confidential
and
// proprietary property of Mathieu Dubois and the possession or use of
this
// file requires a written license from Mathieu Dubois.
//-------------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------------
---------

#include <systemc.h>
#include "bloc.h"       // importation du type Packet

#ifndef LOCALVAR_H
#define LOCALVAR_H

SC_MODULE(localvar)
{

       public:

      // Le bloc pixel venant de la memoire
           sc_port<sc_fifo_in_if<bloc> >in;
           sc_port<sc_fifo_in_if<bloc> >in2;
        //Sortie de la variance
            sc_port<sc_fifo_out_if<bloc> >out;
            sc_port<sc_fifo_out_if<bloc> >out2;

      // Paquet local au module pkt_terminator
                bloc donnee;
           bloc lmean;
           unsigned int row;
           unsigned int col;
      // Ce fonctiontrealise le traitement de localmean
      void traitement(void);
        bloc filter2D(unsigned int ,bloc);
      SC_CTOR(localvar)
```

```
        {
                SC_THREAD(traitement);
        }
};

#endif
```

```
//-------------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------------
--------
// File          : filter.cpp
// Author        : Mathieu Dubois & Serge Catudal
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois & Serge Catudal. This model is
the confidential and
// proprietary property of Mathieu Dubois & Serge Catudal and the
possession or use of this
// file requires a written license from Mathieu Dubois & Serge Catudal.
//-------------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------------
---------

#include "filter.h"

void filter::traitement()
{
      while(true)
      {
                pim->read(g);
                plm->read(local_mean);
                plv->read(local_variance);
                pnoise->read(noise);

cout<<"g : "<<g<<endl;
cout<<"lm: "<<local_mean<<endl;
cout<<"lv: "<<local_variance<<endl;



cout<<endl<<"noise: "<<noise<<endl;

   ptr=0;
   f=g;

   image_filtered = new int[g.image_height*g.image_width];
  for(row=0; row<g.image_height; row++)
    for(col=0; col<g.image_height; col++){
      // f = g - local_mean
```

```
      f.my_image[row][col] = g.my_image[row][col] -
local_mean.my_image[row][col];
      if(row==20 && col==20) cout<<endl<<"g-localmean"<<f;
      // g = local_variance - noise
      g.my_image[row][col] = local_variance.my_image[row][col] - noise;
      if(row==20 && col==20) cout<<endl<<"local_variance-noise: "<<g;
         // g = max(g, 0);
      if(g.my_image[row][col] < 0)
        g.my_image[row][col] = 0;
if(row==20 && col==20) cout<<endl<<"max(g,0) "<<g;

      // local_variance = max(local_variance, noise)
      if(local_variance.my_image[row][col] < noise)
        local_variance.my_image[row][col] = noise;


      // f = f ./ local_variance
      f.my_image[row][col] = f.my_image[row][col] /
local_variance.my_image[row][col];
      // f = f .* g;
      f.my_image[row][col] = f.my_image[row][col] *
g.my_image[row][col];
      // f = f + local_mean
      f.my_image[row][col] = f.my_image[row][col] +
local_mean.my_image[row][col];
      // Conversion de la matrix f en matrix image
      //image_filtered[ptr] = (int)f.my_image[row][col];
      ptr++;
    }
     // f.my_image = image_filtered;
cout<<"filtrer"<<f.my_image[20][20]<<endl;

  out->write(f);




      }
}
```

```
//-------------------------------------------------------------------
--------
// Title         : Wiener filter
// Project       : Video Processing Module
//-------------------------------------------------------------------
--------
// File          : image.cpp
// Author        : Mathieu Dubois & Serge Catudal
// Created       : 10.07.2003
// Last modified : 11.07.2003
//-------------------------------------------------------------------
--------
// Description   :
//-------------------------------------------------------------------
--------
// Copyright (c) 2003 by Mathieu Dubois & Serge Catudal. This model is
the confidential and
// proprietary property of Mathieu Dubois & Serge Catudal and the
possession or use of this
// file requires a written license from Mathieu Dubois & Serge Catudal.
//-------------------------------------------------------------------
---------
// Modification history :
// 10.07.2003 : created
//-------------------------------------------------------------------
---------


#include "image.h"

void image::entree()
{
unsigned int row;
unsigned int col;
int  ptr=0;

      while(true)
      {
read_raw_pgm_file("noise.pgm",VPM_ORIGINAL_IMAGE);

im_entree.my_image = new double*[image_height];
  for(row=0; row<image_height; row++){
    im_entree.my_image[row] = new double[image_width];
    for(col=0; col<image_width; col++){
      im_entree.my_image[row][col] = (double)original_image[ptr];
      ptr++;
}}
                im_entree.image_height=image_height;
                im_entree.image_width=image_width;
                im_entree.nhood=7;

cout<<im_entree;
```

```
                out->write(im_entree);
                out2->write(im_entree);
                out3->write(im_entree);
                wait();
        }
}

void image::sortie()
{
unsigned int row;
unsigned int col;
int ptr=0;

        while(true)
        {

                in->read(im_sortie);

cout<<im_sortie<<endl;
for(row=0; row<image_height; row++)
    for(col=0; col<image_height; col++){
      filtered_image[ptr] = (int)im_sortie.my_image[row][col];
      ptr++;
    }

                write_raw_pgm_file("filtre.pgm",VPM_FILTERED_IMAGE);
                wait();
        }
}

/**
 * @brief Read a Raw PGM image file
 */
void image::read_raw_pgm_file(string filename, vpm_image_type my_type){
  ifstream LireBin;

  char          tampon;
  char*         num = new char[3];
  unsigned int* dim = new unsigned int[3];
  unsigned int  itr_x;
  unsigned int  itr_y;

  unsigned int img_size;
 img_size=255;


  // Read raw PGM image file
  LireBin.open(filename.c_str(),ios::binary);
  LireBin.seekg(0,ios::beg);

  // Read raw PGM image file header
  for(itr_x=0; itr_x<3; itr_x++)
    LireBin.read((char *) &tampon, sizeof(char));
  for(itr_x=0; itr_x<3; itr_x++){
```

```
  itr_y = 0;
  LireBin.read((char *) &tampon, sizeof(char));
  while(itr_y != 3){
    num[itr_y] = tampon;
    LireBin.read((char *) &tampon, sizeof(char));
    itr_y++;
  }
  dim[itr_x] = atoi(num);
}

// Read raw PGM image file pixel
unsigned char num_tmp;
switch(my_type){
case VPM_ORIGINAL_IMAGE:
  {
    original_image = new int[dim[0]*dim[1]];
    noisy_image    = new int[dim[0]*dim[1]];
    filtered_image = new int[dim[0]*dim[1]];
    image_height   = dim[0];
    image_width    = dim[1];
    for(itr_x=0; itr_x<(dim[0]*dim[1]); itr_x++){
      LireBin.read((char *) &num_tmp, sizeof(unsigned char));
      original_image[itr_x] = (int)num_tmp;
      noisy_image[itr_x]    = (int)num_tmp;
      filtered_image[itr_x] = (int)num_tmp;
    }
  }
  break;
case VPM_NOISY_IMAGE:
  {
    noisy_image  = new int[dim[0]*dim[1]];
    image_height = dim[0];
    image_width  = dim[1];
    for(itr_x=0; itr_x<(dim[0]*dim[1]); itr_x++){
      LireBin.read((char *) &num_tmp, sizeof(unsigned char));
      noisy_image[itr_x] = (int)num_tmp;
    }
  }
  break;
case VPM_FILTERED_IMAGE:
  {
    filtered_image = new int[dim[0]*dim[1]];
    image_height   = dim[0];
    image_width    = dim[1];
    for(itr_x=0; itr_x<(dim[0]*dim[1]); itr_x++){
      LireBin.read((char *) &num_tmp, sizeof(unsigned char));
      filtered_image[itr_x] = (int)num_tmp;
    }
  }
  break;
default:
  break;
}
LireBin.close();
```

```
  delete num;
}

/**
 * @brief Write a Raw PGM image file
 */
void image::write_raw_pgm_file(string filename, vpm_image_type
my_type){
  ofstream EcrireBin;

  char* ctmp = new char[10];
  unsigned int  itr;

  // Create raw PGM image file header
  char* header_tmp = new char[20];
  strcat(header_tmp,"P5\n");
  sprintf(ctmp,"%d",image_width);
  strcat(header_tmp,ctmp);
  strcat(header_tmp," ");
  sprintf(ctmp,"%d",image_height);
  strcat(header_tmp,ctmp);
  strcat(header_tmp,"\n255\n");

  // Write raw PGM image file pixel
  EcrireBin.open(filename.c_str(), ios::binary);
  EcrireBin.seekp(0,ios::beg);
  for(itr=0; itr<strlen(header_tmp); itr++)
    EcrireBin.write((char *) &header_tmp[itr],sizeof(char));
  switch(my_type) {
  case VPM_ORIGINAL_IMAGE:
    for(itr=0; itr<(image_width*image_height); itr++)
      EcrireBin.write((char *) &(unsigned
char)original_image[itr],sizeof(unsigned char));
    break;
  case VPM_NOISY_IMAGE:
    for(itr=0; itr<(image_width*image_height); itr++)
      EcrireBin.write((char *) &(unsigned
char)noisy_image[itr],sizeof(unsigned char));
    break;
  case VPM_FILTERED_IMAGE:
    for(itr=0; itr<(image_width*image_height); itr++)
      EcrireBin.write((char *) &(unsigned
char)filtered_image[itr],sizeof(unsigned char));
    break;
  default:
    break;
  }
  EcrireBin.close();

  delete ctmp;
}
```

```
/*Mathieu Dubois code du ARM 2003*/
/* defines */
#define AMBAC (unsigned long*) 0x02000000

unsigned int my_image[255][255];
unsigned int f_my_image[255][255];
unsigned int local_mean_my_image[255][255];
unsigned int local_var_my_image[255][255];
unsigned int g[255][255];
unsigned int row,col;
unsigned int noise;
unsigned int nhood=3;
int image_width =255;
int image_height=255;
unsigned int m_filter[3][3];

/* Prototypes des fonctions */
void irq_handler();


void localmean()
{

  int          ext;
  int          l,r,u,d;
  int          m,n;

  // Initialisation du filtre 2D
  ext = (nhood - 1) / 2;
  for(row=0; row<nhood; row++){
    for(col=0; col<nhood; col++)
      m_filter[row][col] = 1;
  }

  // Initialisation de la matrice g et result
  for(row=0; row<image_height; row++){
    for(col=0; col<image_width; col++)
      g[row][col] = my_image[row][col];

}

  // Filtrage de la matrice temporaire
  for(row=0; row<image_height; row++)
    for(col=0; col<image_width; col++){
      // Determination des bornes utilisables de la matrice de filtrage
      if(row == 0){
        u = ext; d = nhood;
      } else if(row == 1 && (nhood == 5 || nhood == 7)){
        u = ext - 1; d = nhood;
      } else if(row == 2 && nhood == 7){
        u = ext - 2; d =nhood;
      } else if(row == (image_height - 3) && nhood == 7){
        u = 0; d = nhood - ext + 2;
```

```
  }else if(row == (image_height - 2) && (nhood == 5 || nhood ==7)){
        u = 0; d = nhood - ext + 1;
      } else if(row == (image_height - 1)){
        u = 0; d = nhood - ext;
      } else {
        u = 0; d = nhood;
      }

     if(col == 0){
        l = ext; r = nhood;
      } else if(col == 1 && (nhood == 5 || nhood == 7)){
        l = ext - 1; r = nhood;
      } else if(col == 2 && nhood == 7){
        l = ext - 2; r = nhood;
      } else if(col == (image_width - 3) && nhood == 7){
        l = 0; r = nhood - ext + 2;
  } else if(col == (image_width - 2) && (nhood == 5 || nhood ==7)){
        l = 0; r = nhood - ext + 1;
      } else if(col == (image_width - 1)){
        l = 0; r = nhood - ext;
      } else {
        l = 0; r = nhood;
      }

     // Apply 2D filter
     local_mean_my_image[row][col] = 0;
     for(m=u-ext; m<d-ext; m++)
       for(n=l-ext; n<r-ext; n++)
         local_mean_my_image[row][col] = local_mean_my_image[row][col]
+ (g[row+m][col+n] * m_filter[m+ext][n+ext]);
         local_mean_my_image[row][col] = local_mean_my_image[row][col]
/ (nhood * nhood);
     }

  }

void localvar()
{

  int          ext;

  int          l,r,u,d;
  int          m,n;

  // Initialisation du filtre 2D
  ext = (nhood - 1) / 2;
  for(row=0; row<nhood; row++){
    for(col=0; col<nhood; col++)
      m_filter[row][col] = 1;
  }

  // Initialisation de la matrice g et result
  for(row=0; row<image_height; row++){
    for(col=0; col<image_width; col++)
```

```
      g[row][col] = my_image[row][col]*my_image[row][col];

}

  // Filtrage de la matrice temporaire
  for(row=0; row<image_height; row++)
    for(col=0; col<image_width; col++){
      // Determination des bornes utilisables de la matrice de filtrage
      if(row == 0){
        u = ext; d = nhood;
      } else if(row == 1 && (nhood == 5 || nhood == 7)){
        u = ext - 1; d = nhood;
      } else if(row == 2 && nhood == 7){
        u = ext - 2; d =nhood;
      } else if(row == (image_height - 3) && nhood == 7){
        u = 0; d = nhood - ext + 2;
  }else if(row == (image_height - 2) && (nhood == 5 || nhood ==7)){
        u = 0; d = nhood - ext + 1;
      } else if(row == (image_height - 1)){
        u = 0; d = nhood - ext;
      } else {
        u = 0; d = nhood;
      }

     if(col == 0){
        l = ext; r = nhood;
      } else if(col == 1 && (nhood == 5 || nhood == 7)){
        l = ext - 1; r = nhood;
      } else if(col == 2 && nhood == 7){
        l = ext - 2; r = nhood;
      } else if(col == (image_width - 3) && nhood == 7){
        l = 0; r = nhood - ext + 2;
  } else if(col == (image_width - 2) && (nhood == 5 || nhood ==7)){
        l = 0; r = nhood - ext + 1;
      } else if(col == (image_width - 1)){
        l = 0; r = nhood - ext;
      } else {
        l = 0; r = nhood;
      }

      // Apply 2D filter
      local_var_my_image[row][col] = 0;
      for(m=u-ext; m<d-ext; m++)
        for(n=l-ext; n<r-ext; n++)
          local_var_my_image[row][col] = local_var_my_image[row][col] +
(g[row+m][col+n] * m_filter[m+ext][n+ext]);
          local_var_my_image[row][col] = local_var_my_image[row][col] /
(nhood * nhood);
    }

for(row=0; row<image_height; row++)
    for(col=0; col<image_width; col++)
  local_var_my_image[row][col] = local_var_my_image[row][col] -
local_mean_my_image[row][col]*local_mean_my_image[row][col];
```

```
  }
void mean(void)
{
 for(row=0; row<image_height; row++)
    for(col=0; col<image_width; col++)
    noise = noise + my_image[row][col];
    noise = noise / (image_height * image_width);


 }
void filter(void)
{

      for(row=0; row<image_height; row++)
    for(col=0; col<image_height; col++){
      // f = g - local_mean
      f_my_image[row][col] = my_image[row][col] -
local_mean_my_image[row][col];
      // g = local_variance - noise
      g[row][col] = local_var_my_image[row][col] - noise;
       // g = max(g, 0);
      if(g[row][col] < 0)
        g[row][col] = 0;

      // local_variance = max(local_variance, noise)
      if(local_var_my_image[row][col] < noise)
        local_var_my_image[row][col] = noise;


      // f = f ./ local_variance
      f_my_image[row][col] = f_my_image[row][col] /
local_var_my_image[row][col];
      // f = f .* g;
      f_my_image[row][col] = f_my_image[row][col] * g[row][col];
      // f = f + local_mean
      f_my_image[row][col] = f_my_image[row][col] +
local_mean_my_image[row][col];
      // Conversion de la matrix f en matrix image
      //image_filtered[ptr] = (int)f.my_image[row][col];

    }
 }

int main(void)
{

localmean();
localvar();
mean();
filter();


  // initialisation de la matrix f
```

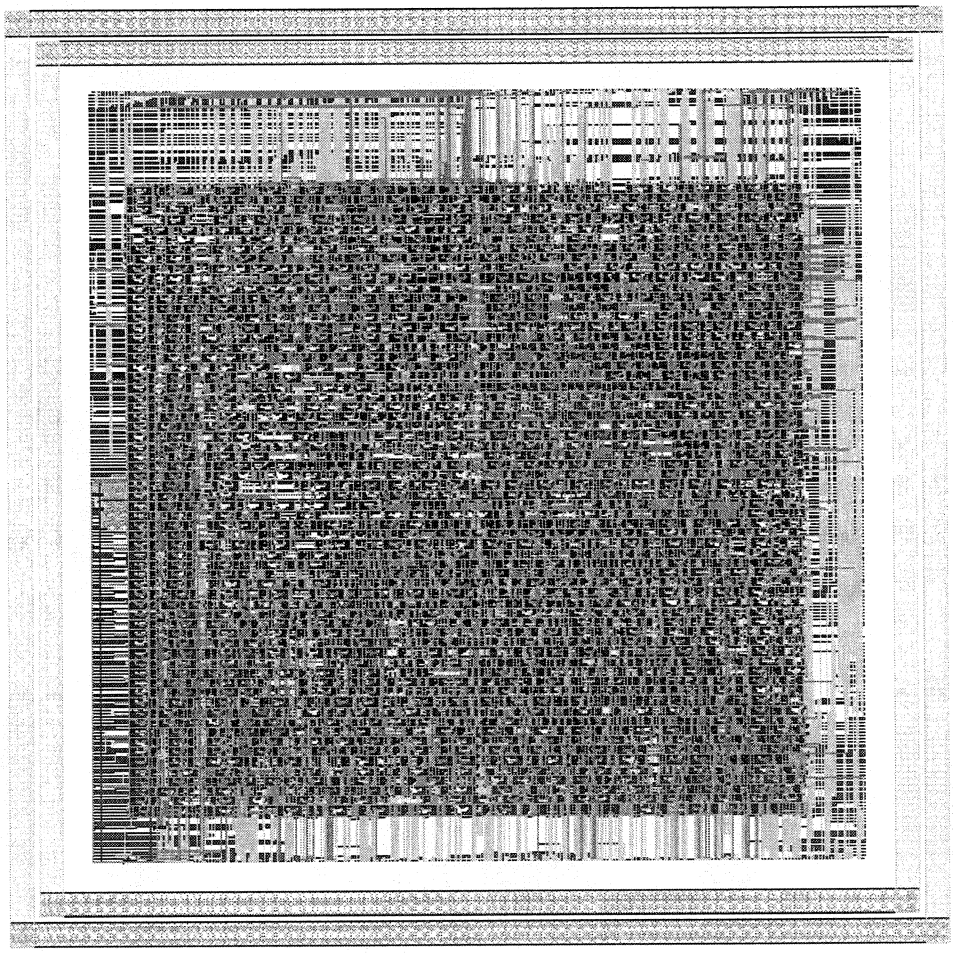
```
while(1);
  return(0);
 }
```

ANNEXE B

Bus de communication

Bus Circulaire

La figure suivante présente le dessin des masques du bus après placement et routage. D'après les analyses de délais statique réalisée sur ce circuit, la fréquence maximale estimée en ne tenant pas compte des capacités parasites est de 510.12 MHz



Dessin des masques du bus 320MHz

L'implantation matérielle du bus AHB 320 MHz exige une surface de 481.8µm X 481.6µm qui correspond à 3413 portes avec la technologie CMOS 0.18 micron..

Analyse de surface pour 8 maitres et un esclave du bus générique

```
****************************************
Report : area
Design : synthesetop_MASTERS8_SLAVES1
Version: 2003.06
Date   : Wed Oct  1 15:43:26 2003
****************************************

Library(s) Used:

   vst_n18_sc_tsm_c4_wc (File:
/CMC/kits/cmosp18.4.3/synopsys/2002/syn/vst_n18_sc_tsm_c4_wc.db)

Number of ports:          1004
Number of nets:           5823
Number of cells:          5293
Number of references:       74

Combinational area:      74225.187500
Noncombinational area:   103610.843750
Net Interconnect area:     undefined  (No wire load specified)

Total cell area:         177837.796875
Total area:              undefined
```

Exemple d'analyse fréquentielle du bus générique

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : synthesetop_MASTERS8_SLAVES1
Version: 2003.06
Date   : Wed Oct  1 15:43:26 2003
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: I7_6/memoire_reg[2]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: I5/addr_int_reg[5]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  I7_6/memoire_reg[2]/CK (DFFRPB1)         0.00       0.00 r
  I7_6/memoire_reg[2]/QB (DFFRPB1)         0.26       0.26 f
  U5014/Z (EXNOR2D1)                       0.26       0.52 r
  U5015/Z (NAN3D1)                         0.08       0.60 f
  U5022/Z (MUXB2D0)                        0.25       0.85 r
  U5020/Z (EXNOR2D2)                       0.38       1.24 f
  U4929/Z (NAN2D2)                         0.10       1.33 r
  U4979/Z (BUFDA)                          0.13       1.47 r
  U5403/Z (NOR2D2)                         0.06       1.53 f
  U5889/Z (NOR2D4)                         0.07       1.60 r
  U5890/Z (NAN4M1D2)                       0.10       1.70 f
  U5482/Z (NAN3M1D2)                       0.16       1.86 f
  I5/addr_int_reg[5]/D (DFFRPQ1)           0.00       1.86 f
  data arrival time                                   1.86

  clock clk (rise edge)                    1.12       1.12
```

clock network delay (ideal)	0.00	1.12
I5/addr_int_reg[5]/CK (DFFRPQ1)	0.00	1.12 r
library setup time	-0.15	0.97
data required time		0.97
---		
data required time	0.97	
data arrival time	-1.86	
---		
slack (VIOLATED)	-0.89	

Fichiers principaux	Description des fichiers
adaptateur_adaptateur.vhd	Adaptateur générique AMBA
ahbarbiterdecoder_rtl.vhd	Arbitre et décodeur d'adresse
ahbmaster1_ahbmaster1.vhd	Émulateur Maître
amba_pkg.vhd	Fichier de définition de la norme AMBA
arbiter_arbiter.vhd	L'arbitre par encodage de priorité du bus AMBA
clocker_behav.vhd(1)	L'horloge du système
decodeur_decodeur.vhd	Décodeur d'adresse
home_pkg.vhd	Définition des vecteurs de test
matrix_interconnect.vhd	Matrice d'interconnexion
mem_1r_1w_mem_1r_1w_rtl.vhd	Module Mémoire
ptslv_pont.vhd	Adaptateur esclave
registre_registres.vhd	Création des registres pour les tests
top_struct.vhd (1)	Schéma au niveau du système
total.csh	Script de synthèse pour la vitesse du bus max
synthesetop_struct	Schéma au niveau du système pour la synthèse
gen_total.csh	Générateur de possibilité des domaines d'horloge
top_struct.vhd(2)	Modèle pour générer un top automatique des domaines d'horloge différents , controllé par gen_total.csh
clocker_behav.vhd(2)	Modèle pour générer des horloges, controllé par gen_total.csh

```
-------------------------------------------------------------------------
--------
-- Titre         : Adaptateur générique
-- Projet        : Bus générique
-------------------------------------------------------------------------
--------
-- Fichier               : adaptateur_adaptateur.vhd
-- Auteur(s)             : Mathieu Dubois
-- Création              : 3/8/2003
-- Dernière modification : 3/8/2003
-------------------------------------------------------------------------
--------
-- Description   : Pont AHB/AHB de base
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois. La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------


LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY dynamic;
USE dynamic.AMBA.ALL;


ENTITY adaptateur IS
   PORT(
      ahbmstin  : IN     ahb_mst_in_type;
      ahbslvin  : IN     ahb_mst_out_type;
      clk       : IN     std_logic;
      reset_n   : IN     std_logic;
      ahbmstout : OUT    ahb_mst_out_type;
      ahbslvout : OUT    ahb_mst_in_type;
      clkbus    : IN     std_logic
   );

-- Declarations

END adaptateur ;

-- hds interface_end
ARCHITECTURE adaptateur OF adaptateur IS
```

```
-------------------------------------------------------------------------
---------
signal etat          : std_logic_vector(1 downto 0);  -- etat
signal data         : std_logic_vector(31 downto 0);  -- Module is
selected with valid transfer signal
adr,read_data,read_data2,tmpwritehaddr,
tmpwritehwdata,tmpwritehaddr2,memoiredata, read_data_mem,read_data_tmp
: std_logic_vector(31 downto 0);
-- Module was selected with valid transfer
signal write_tmp,trans_tmp,resp_tmp
,trans_s,trans_s2,attentetmp,bardata,control,controlmem,
control2,ctrl_memoire2 ,ctrl_read_data: std_logic;
-- Module was selected with valid transfer
signal req,check,tmpwritehwrite,tmpwritehwrite2,requetebus,
memoirehgrant,memoirereq,memoirereq2,tue,tue2,memmoirehwrite
: std_logic;
signal ack,memoire2,mem_trans,t ,memo,sign_memoire2,memoire2_tmp :
std_logic_vector(1 downto 0);
signal registre,memoire,registretmp,mm2,mm2jk,memoirejk :
std_logic_vector(2 downto 0);
type t_amba is
(IDLE,ADRESSE,DATA_S,GRANT,mode_burst,attente,memoire_data,init);

Signal present_etat_amba, next_etat_amba : t_amba;
Signal present_etat_amba_bus, next_etat_amba_bus: t_amba;
-------------------------------------------------------------------------
---------
-- Beginning of main code
-------------------------------------------------------------------------
---------
begin
               ahbmstout.hsize<="000";
               ahbmstout.hprot<="0000";
               ahbmstout.hlock<='0';
               ahbmstout.hburst<="000";

ahbmstout.hbusreq<= (req ) and requetebus ;
t<=ahbslvin.HTRANS;

--Temporisation pour l'ecriture
writetmp:process (reset_n , clk)
begin
 if (reset_n  = '0') then
        tmpwritehaddr<=(others=>'0');
       tmpwritehwdata<=(others=>'0');
       tmpwritehwrite<='0';
         tmpwritehwrite2<='0';
       tmpwritehaddr2<=(others=>'0');

elsif (clk'event and clk = '1') then

       tmpwritehaddr<=ahbslvin.haddr;
       tmpwritehwdata<=ahbslvin.HWDATA;
       tmpwritehwrite<=ahbslvin.hwrite;
```

```
        tmpwritehwrite2<=tmpwritehwrite;
      tmpwritehaddr2<=tmpwritehaddr;
end if;

end process;

--Machine a etat du module synchrone

syn:process (reset_n , clk)
begin
 if (reset_n  = '0') then
      present_etat_amba <= IDLE;
 elsif (clk'event and clk = '1') then
      present_etat_amba <= next_etat_amba;
 end if;

end process;

--Machine à état du module
  etat_amba_1 : process (present_etat_amba,ahbslvin,mem_trans)
  begin
    case present_etat_amba is

       when IDLE=>
            --detection d'une demande non sequentiel
            if ahbslvin.hBUSREQ='1'  then
               next_etat_amba <=ADRESSE;
             req<='0';
            else
               next_etat_amba <=IDLE;
              req<='0';
            end if;

       when ADRESSE =>

      --Niveau pipeline 1
      if ahbslvin.HTRANS /= HTRANS_IDLE  then
      --detection d'une ecriture
       if ahbslvin.HWRITE='1'  then

             req<='0';
             next_etat_amba <=DATA_S;

       --detection d'une lecture
       else
            req<='1';
            next_etat_amba <=ADRESSE;
        end if;

       else
       next_etat_amba <=IDLE;
         req<='0';

       end if;
```

```
       when DATA_S =>

       if mem_trans /= HTRANS_IDLE  then
       --Detection d'une ecriture

             next_etat_amba<=DATA_S;
             req<='1';

       else
             next_etat_amba<=IDLE;
             req<='0';
       end if;

       when OTHERS=>
            next_etat_amba<=IDLE;
            req<='0';

     end case;

  end process etat_amba_l;

--memoirisation de htrans
registre_htrans: process(reset_n , clk)
begin

 if (reset_n  = '0') then
             mem_trans<="00";
    elsif (clk'event and clk = '1') then
             mem_trans <=ahbslvin.HTRANS;
    end if;
end process registre_htrans;

--registre a decalage
registre_decalage: process(reset_n , clk)
begin

 if (reset_n  = '0') then
    registre<="100";
    elsif (clk'event and clk = '1') then
    registre<=registre(0) & registre(2) & registre(1);
    end if;

end process registre_decalage;


--gestion de la reponse
signal_ready: process(reset_n , clk)
begin

 if (reset_n  = '0') then
    ahbslvout.HRESP<="00";
    elsif (clk'event and clk = '1') then
```

```
     case next_etat_amba is

       when IDLE =>
             ahbslvout.HRESP<="00";
       when ADRESSE =>
           ahbslvout.HRESP<="00";
       when DATA_S=>
           ahbslvout.HRESP<="10";
       when OTHERS=> NULL;

     end case;
    end if;
end process signal_ready;

--machine a etat du bus synchrone
syn_bus:process (reset_n , clkbus)
begin
 if (reset_n  = '0') then
      present_etat_amba_bus <= init;

elsif (clkbus'event and clkbus = '1') then
 present_etat_amba_bus <= next_etat_amba_bus;
end if;

end process;
--machine a etat du bus
comb_etat_amba_lbus : process
(present_etat_amba_bus,ahbmstin,req,registretmp,registre,present_etat_a
mba,memoirereq,memoirereq2,memoire,reset_n)
  begin

 case present_etat_amba_bus is

       when IDLE=>

--detection d'une requête
            if req='1' or memoirereq='1' or memoirereq2='1' then
              if ahbmstin.hgrant='1'  then
                next_etat_amba_bus <=ADRESSE;
                requetebus<='0';
              else
                next_etat_amba_bus <=idle;
                requetebus<='1';
              end if;
           else
                requetebus<='1';
                attentetmp<='0';
                next_etat_amba_bus <=IDLE;
           end if;
--initialiser les signaux
      sign_memoire2<="00";
      read_data_tmp<=(others=>'0');
      ctrl_memoire2<='1';
      ctrl_read_data<='1';
```

```
      ahbmstout.htrans<="00";
      ahbmstout.hWDATA<=(others=>'0');
      ahbmstout.hADDR<=(others=>'0');
      ahbmstout.hWRITE<='0';

       when ADRESSE =>

       ahbmstout.hWDATA<=(others=>'0');
       next_etat_amba_bus <=DATA_S;
       ahbmstout.htrans<="10";
       requetebus<='0';

--detection du niveau de pipeline
if control2='1' then

      if present_etat_amba=DATA_S then
      --mode ecriture
      if memoire=registre then
        ahbmstout.hADDR<=tmpwritehaddr;
        ahbmstout.hWRITE<=tmpwritehwrite;
       else
        ahbmstout.hADDR<=tmpwritehaddr2;
        ahbmstout.hWRITE<=tmpwritehwrite2;
      end if;

      else



      --mode lecture
      if memoire=registre then
        ahbmstout.hADDR<=ahbslvin.haddr;
        ahbmstout.hWRITE<=ahbslvin.hwrite;
       else
        ahbmstout.hADDR<=tmpwritehaddr;
        ahbmstout.hWRITE<=tmpwritehwrite;
      end if;

      end if;
else
         if present_etat_amba=DATA_S then
      --mode ecriture
      if mm2=registre then
        ahbmstout.hADDR<=tmpwritehaddr;
        ahbmstout.hWRITE<=tmpwritehwrite;
       else
        ahbmstout.hADDR<=tmpwritehaddr2;
        ahbmstout.hWRITE<=tmpwritehwrite2;
      end if;

      else
      --mode lecture
      if mm2=registre then
        ahbmstout.hADDR<=ahbslvin.haddr;
```

```
        ahbmstout.hWRITE<=ahbslvin.hwrite;
       else
        ahbmstout.hADDR<=tmpwritehaddr;
        ahbmstout.hWRITE<=tmpwritehwrite;
      end if;

      end if;
end if;

      ctrl_memoire2<='1';
      sign_memoire2<="00";
      read_data_tmp<=(others=>'0');
      ctrl_read_data<='1';

      when DATA_S =>

      ahbmstout.htrans<="00";
      next_etat_amba_bus <=memoire_data;
      ahbmstout.hADDR<=(others=>'0');
      ahbmstout.hWRITE<='0';
      requetebus<='0';

      --detection du niveau de pipeline
        if control2='0' then
         if memoire=registre then
         ahbmstout.hWDATA<=ahbslvin.HWDATA;
         --- control<='1';
         else
         ahbmstout.hWDATA<=tmpwritehwdata;
         -- control<='1';
         end if;

        else

         if mm2=registre then
            ahbmstout.hWDATA<=ahbslvin.HWDATA;
           else
            ahbmstout.hWDATA<=tmpwritehwdata;
         end if;
        end if;

       ctrl_memoire2<='1';
       sign_memoire2<="00";
       read_data_tmp<=(others=>'0');
       ctrl_read_data<='1';
       ahbmstout.hADDR<=(others=>'0');
       ahbmstout.hWRITE<='0';

      when memoire_data =>

      ahbmstout.htrans<="00";
      ahbmstout.hWDATA<=(others=>'0');
      ahbmstout.hADDR<=(others=>'0');
      ahbmstout.hWRITE<='0';
```

```
      read_data_tmp<=ahbmstin.hrdata;
      ctrl_read_data<='0';
--detection du niveau de pipeline
      if control2='0' then
           if memoire=registre then
                next_etat_amba_bus <=attente;
                sign_memoire2<="10" ;--sert a avancer l'horloge
                requetebus<='0';
           else
                next_etat_amba_bus <=IDLE;
                sign_memoire2<="00";
                requetebus<='1';
           end if;
       else
           if mm2=registre then
                next_etat_amba_bus <=attente;
                sign_memoire2<="10";
                requetebus<='0';
           else
                next_etat_amba_bus <=IDLE;
                sign_memoire2<="00";
                requetebus<='1';
           end if;
        end if;
         attentetmp<='1';
         ctrl_memoire2<='0';

        when attente =>

        ctrl_memoire2<='1';
        sign_memoire2<="00";
        read_data_tmp<=(others=>'0');
        ctrl_read_data<='1';
        ahbmstout.htrans<="00";
        ahbmstout.hWDATA<=(others=>'0');
        ahbmstout.hADDR<=(others=>'0');
        ahbmstout.hWRITE<='0';
--detection du niveau de pipeline
if control2='0' then
           if memoire=registre  then
            next_etat_amba_bus <=attente;
            requetebus<='0';
           else
            next_etat_amba_bus <=IDLE;
            requetebus<='1';
           end if;
else

           if mm2=registre then
            next_etat_amba_bus <=attente;
            requetebus<='0';
           else
              next_etat_amba_bus <=IDLE;
            requetebus<='1';
```

```
            end if;

end if;


          when init  =>
            read_data_tmp<=(others=>'0');
            requetebus<='0';
          next_etat_amba_bus <=IDLE;
          sign_memoire2<="00";
            ctrl_memoire2<='1';
            ctrl_read_data<='1';
            ahbmstout.htrans<="00";
            ahbmstout.hWDATA<=(others=>'0');
            ahbmstout.hADDR<=(others=>'0');
            ahbmstout.hWRITE<='0';
        when OTHERS=>
            next_etat_amba_bus <=IDLE;
            requetebus<='0';
            sign_memoire2<="00";
            ctrl_memoire2<='1';
            read_data_tmp<=(others=>'0');
            ctrl_read_data<='1';
            ahbmstout.htrans<="00";
            ahbmstout.hWDATA<=(others=>'0');
            ahbmstout.hADDR<=(others=>'0');
            ahbmstout.hWRITE<='0';
      end case;


  end process comb_etat_amba_lbus;




--multiplexeur l'avancement de la donnée

read_data_mux: process(ctrl_read_data,read_data_tmp,read_data_mem)
begin
if ctrl_read_data='1' then

            read_data<=read_data_mem;
else

            read_data<=read_data_tmp;

end if;
end process read_data_mux;

--temporisation la donnée lu sur le bus

read_data_syn: process(reset_n , clkbus)
```

```
begin
if (reset_n  = '0') then
     read_data_mem<=(others=>'0');

elsif (clkbus'event and clkbus = '1') then
     read_data_mem<=read_data;
end if;

end process read_data_syn;



--memoire controleur
memoire2_mux: process(ctrl_memoire2,memoire2_tmp,sign_memoire2)
begin
if ctrl_memoire2='1' then
      memoire2<=memoire2_tmp;
else
      memoire2<=sign_memoire2;
end if;
end process memoire2_mux;

--memoire
memoire2_syn: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
      memoire2_tmp<="00";

elsif (clkbus'event and clkbus = '1') then
      memoire2_tmp<=memoire2;
end if;

end process memoire2_syn;







--temporise la valeur lu avec le module
memorisedatalent: process(reset_n , clk)
begin
if (reset_n  = '0') then
      read_data2 <= read_data;
elsif (clk'event and clk = '1') then
      read_data2 <= read_data;
end if;
end process memorisedatalent;

--mux de la valeur lu
mux_rdata : process (read_data,memoire2,read_data2)
begin
if memoire2="00" then
```

```
     ahbslvout.HRDATA<=read_data;
else
     ahbslvout.HRDATA<=read_data2;
end if;
end process mux_rdata;

--Generation du signal de controle2
mectrl2: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
      control2 <= '0';

elsif (clkbus'event and clkbus = '1') then

  if present_etat_amba_bus=adresse then
      control2 <= not(control2) ;
  end if;

end if;
end process mectrl2;

--memoire l'authorisation d'acces au bus
memorisegrant: process(reset_n , clk)
begin
if (reset_n  = '0') then
    memoirehgrant   <= '0';
elsif (clk'event and clk = '1') then
    memoirehgrant <= ahbmstin.hgrant;
end if;
end process memorisegrant;

--memorise la memoire 1
memont: process(reset_n , clkbus)
begin

if (reset_n  = '0') then
    memoire   <= (others=>'0');
elsif (clkbus'event and clkbus = '1') then
   memoire <= memoirejk ;
end if;

end process memont;

--memorise la memoire 2
memoriseme: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
    mm2   <= (others=>'0');
elsif (clkbus'event and clkbus = '1') then
    mm2 <= mm2jk;
end if;

end process memoriseme;
```

```
--memorise la requete
memorisereq: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
      memoirereq <= '0';

elsif (clkbus'event and clkbus = '1') then
      memoirereq <=req;
end if;
end process memorisereq;

--memoirise le registre
memreq: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
      registretmp <= (others=>'0');

elsif (clkbus'event and clkbus = '1') then
registretmp <=registre;
end if;
end process memreq;

--generation du signal control
memctrl: process(reset_n , clk)
begin
if (reset_n  = '0') then
      control <= '0';

elsif (clk'event and clk = '1') then
  if req='1' then
     control <= not(control);
  end if;
end if;
end process memctrl;
```

```
--generation de tue et tue2
crregistre: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
     memoirejk<=(others=>'0');
     mm2jk  <= (others=>'0');
     tue<= '0';
     tue2<= '0';
```

```
elsif (clkbus'event and clkbus = '1') then
     if req = '1' then

       if control='1' then
            if tue='0' then
              memoirejk<=registre;
               tue<='1';
              else
                memoirejk<=memoirejk;
             end if;
         tue2<='0';

       else
              if tue2='0' then
                mm2jk<=registre;
                tue2<='1';
              else
                mm2jk<=mm2jk;
              end if;

       tue<='0';

       end if;
       end if;
end if;
end process crregistre;

--memoire la requete une deuxieme fois
memoriser2: process(reset_n , clkbus)
begin
if (reset_n  = '0') then
 memoirereq2 <= '0';
elsif (clkbus'event and clkbus = '1') then
 memoirereq2 <=memoirereq;
end if;

end process memoriser2;

END adaptateur;
```

```
------------------------------------------------------------------------
--------
-- Titre         : Struture de bus AHB AMBA
-- Projet        : Bus generique
------------------------------------------------------------------------
--------
-- Fichier               : ahbarbiterdecoder_rtl.vhd
-- Auteur(s)             : Mathieu Dubois
-- Création             : 1/8/2003
-- Dernière modification :4/8/2003
------------------------------------------------------------------------
--------
-- Description   : Bus  AHB
------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois.La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Matheu
uDubois.
------------------------------------------------------------------------
--------
-- L'historique de modification :
--
------------------------------------------------------------------------
--------
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.all;
USE IEEE.Std_Logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;
LIBRARY std;
USE std.textio.ALL;


ENTITY AHBArbiterDecoder IS
   GENERIC(
      MASTERS : integer := 16;
      SLAVES  : integer := 16
   );
   --  number of masters
   --  number of slaves
   --
   PORT(
      hclk        : IN     Std_ULogic;
      hresetn     : IN     Std_ULogic;
      ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
      ahbarbtomst : OUT    ahb_mst_in_vector (0 TO MASTERS-1);
      ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
      ahbarbtoslv : OUT    ahb_slv_in_vector (0 TO SLAVES-1)
   );
```

```
-- Declarations

END AHBArbiterDecoder ;

ARCHITECTURE RTL OF AHBArbiterDecoder IS

   -- Architecture declarations

   -- Internal signal declarations
   SIGNAL hsel_d      : unsigned(SLAVES-1 DOWNTO 0);
   SIGNAL haddr_d     : std_logic_vector(31 DOWNTO 0);
   SIGNAL hmastlock_a : std_logic;
   SIGNAL hmaster_a   : unsigned(3 DOWNTO 0);
   SIGNAL haddr_a     : std_logic_vector(31 DOWNTO 0);
   SIGNAL htrans_a    : std_logic_vector(1 DOWNTO 0);
   SIGNAL hburst_a    : std_logic_vector(2 DOWNTO 0);
   SIGNAL hsplit_a    : std_logic_vector(15 DOWNTO 0);
   SIGNAL hresp_a     : std_logic_vector(1 DOWNTO 0);
   SIGNAL hready_a    : std_logic;
   SIGNAL hbusreq     : unsigned(MASTERS-1 DOWNTO 0);
   SIGNAL hgrant      : unsigned(MASTERS-1 DOWNTO 0);
   SIGNAL hlock       : unsigned(MASTERS-1 DOWNTO 0);


   -- Component Declarations
   COMPONENT arbiter
   GENERIC (
      masters : integer := 8
   );
   PORT (
      hresetn     : IN     Std_ULogic ;
      hclk        : IN     Std_ULogic ;
      hbusreq     : IN     unsigned (MASTERS-1 DOWNTO 0);
      hgrant      : OUT    unsigned (MASTERS-1 DOWNTO 0);
      haddr_a     : IN     std_logic_vector (31 DOWNTO 0);
      htrans_a    : IN     std_logic_vector (1 DOWNTO 0);
      hburst_a    : IN     std_logic_vector (2 DOWNTO 0);
      hsplit_a    : IN     std_logic_vector (15 DOWNTO 0);
      hresp_a     : IN     std_logic_vector (1 DOWNTO 0);
      hready_a    : IN     std_logic ;
      hmaster_a   : OUT    unsigned (3 DOWNTO 0);
      hmastlock_a : OUT    std_logic ;
      hlock       : IN     unsigned (MASTERS-1 DOWNTO 0)
   );
   END COMPONENT;
   COMPONENT decodeur
   GENERIC (
      slaves : integer := 16
   );
   PORT (
      haddr_d : IN     std_logic_vector (31 DOWNTO 0);
      hsel_d  : OUT    unsigned (SLAVES-1 DOWNTO 0)
   );
```

```
   END COMPONENT;
   COMPONENT matrix
   GENERIC (
      masters : integer := 16;
      slaves  : integer := 16
   );
   PORT (
      ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
      ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
      hclk        : IN     Std_ULogic ;
      hresetn     : IN     Std_ULogic ;
      hlock       : OUT    unsigned (MASTERS-1 DOWNTO 0);
      hbusreq     : OUT    unsigned (MASTERS-1 DOWNTO 0);
      haddr_d     : OUT    std_logic_vector (31 DOWNTO 0);
      haddr_a     : OUT    std_logic_vector (31 DOWNTO 0);
      htrans_a    : OUT    std_logic_vector (1 DOWNTO 0);
      hburst_a    : OUT    std_logic_vector (2 DOWNTO 0);
      hsplit_a    : OUT    std_logic_vector (15 DOWNTO 0);
      hresp_a     : OUT    std_logic_vector (1 DOWNTO 0);
      hready_a    : OUT    std_logic ;
      hmaster_a   : IN     unsigned (3 DOWNTO 0);
      hmastlock_a : IN     std_logic ;
      hsel_d      : IN     unsigned (SLAVES-1 DOWNTO 0);
      ahbarbtoslv : OUT    ahb_slv_in_vector (0 TO SLAVES-1);
      ahbarbtomst : OUT    ahb_mst_in_vector (0 TO MASTERS-1);
      hgrant      : IN     unsigned (MASTERS-1 DOWNTO 0)
   );
   END COMPONENT;

   -- Optional embedded configurations
   -- pragma synthesis_off
   FOR ALL : arbiter USE ENTITY dynamic.arbiter;
   FOR ALL : decodeur USE ENTITY dynamic.decodeur;
   FOR ALL : matrix USE ENTITY dynamic.matrix;
   -- pragma synthesis_on


BEGIN
   -- Instance port mappings.
   I0 : arbiter
      GENERIC MAP (
         masters => MASTERS
      )
      PORT MAP (
         hresetn     => hresetn,
         hclk        => hclk,
         hbusreq     => hbusreq,
         hgrant      => hgrant,
         haddr_a     => haddr_a,
         htrans_a    => htrans_a,
         hburst_a    => hburst_a,
         hsplit_a    => hsplit_a,
         hresp_a     => hresp_a,
         hready_a    => hready_a,
```

```
      hmaster_a   => hmaster_a,
      hmastlock_a => hmastlock_a,
      hlock       => hlock
   );
I1 : decodeur
   GENERIC MAP (
      SLAVES => SLAVES
   )
   PORT MAP (
      haddr_d => haddr_d,
      hsel_d  => hsel_d
   );




I2 : matrix
   GENERIC MAP (
      masters => MASTERS,
      slaves  => SLAVES
   )
   PORT MAP (
      ahbmsttoarb => ahbmsttoarb,
      ahbslvtoarb => ahbslvtoarb,
      hclk        => hclk,
      hresetn     => hresetn,
      hlock       => hlock,
      hbusreq     => hbusreq,
      haddr_d     => haddr_d,
      haddr_a     => haddr_a,
      htrans_a    => htrans_a,
      hburst_a    => hburst_a,
      hsplit_a    => hsplit_a,
      hresp_a     => hresp_a,
      hready_a    => hready_a,
      hmaster_a   => hmaster_a,
      hmastlock_a => hmastlock_a,
      hsel_d      => hsel_d,
      ahbarbtoslv => ahbarbtoslv,
      ahbarbtomst => ahbarbtomst,
      hgrant      => hgrant
   );

END RTL;
```

```
-------------------------------------------------------------------------
--------
-- Titre          : Emulateur maître AHB de base
-- Projet         : Bus générique
-------------------------------------------------------------------------
--------
-- Fichier                : ahbmaster1_ahbmaster1.vhd
-- Auteur(s)              : Mathieu Dubois
-- Création               : 5/8/2003
-- Dernière modification  : 5/8/2003
-------------------------------------------------------------------------
--------
-- Description    : Emulateur maître
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois. La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
LIBRARY dynamic;
USE dynamic.AMBA.ALL;
USE dynamic.txt_util.all;

ENTITY AHBMaster1 IS
   PORT(
      ahbmsttoarb : IN     ahb_mst_in_type;
      clk         : IN     std_logic;
      reset_n     : IN     std_logic;
      ahbarbtomst : OUT    ahb_mst_out_type;
      reg_addr    : IN     std_logic_vector (31 DOWNTO 0);
      reg_data    : IN     std_logic_vector (31 DOWNTO 0)
   );

-- Declarations

END AHBMaster1 ;

-- hds interface_end
ARCHITECTURE AHBMaster1 OF AHBMaster1 IS
```

```
signal data : std_logic_vector(31 downto
0):="00000000000000000000000000001001";
signal addr: std_logic_vector(31 downto
0):="00000000000000000000000000001001";
signal regaddr: std_logic_vector(31 downto
0):="00000000000000000000000000100000";
signal regdata: std_logic_vector(31 downto
0):="00000000000000000000000000100000";
signal regaddrmul: std_logic_vector(63 downto 0);
signal regdatamul: std_logic_vector(63 downto 0);
signal ldata : std_logic_vector(31 downto
0):="00000000000000000000000000001001";

signal regdata2: std_logic_vector(31 downto
0):="00000000000000000000000000100000";
signal tmpmul: std_logic_vector(31 downto
0):="00000000000000000000000000001001";
BEGIN

PROCESS
variable data1: std_logic_vector(31 downto
0):="00000000000000000000000000000000";
variable addr1: std_logic_vector(31 downto
0):="00000000000000000000000000000000";
variable tmp: std_logic_vector(31 downto
0):="00000000000000000000000000000000";



BEGIN

--initialisation

ahbarbtomst.HWDATA<="00000000000000000000000000000000";
ahbarbtomst.HADDR<="00000000000000000000000000000000";
ahbarbtomst.HWRITE<='0';
ahbarbtomst.HTRANS<="00";
ahbarbtomst.HBURST<="000";
ahbarbtomst.HBUSREQ<='0';
wait until RESET_n='1';
wait until clk'event and clk='1';
wait until clk'event and clk='1';
wait until clk'event and clk='1';
wait until clk'event and clk='1';

addr1:=addr;
data1:=data;
regaddr<=reg_addr;
regdata<=reg_data;

--ecriture #1
ahbarbtomst.HBUSREQ<='1';
wait until clk'event and clk='1';
ahbarbtomst.HADDR<=reg_addr;
```

```
ahbarbtomst.HWRITE<='1';
ahbarbtomst.HTRANS<="10";
wait until clk'event and clk='1';

ahbarbtomst.HWDATA<= regdata;
ahbarbtomst.HBUSREQ<='0';
ahbarbtomst.HTRANS<="00";
wait until clk'event and clk='1';
wait until clk'event and clk='1';
ahbarbtomst.HWRITE<='0';
ahbarbtomst.HBUSREQ<='1';

wait until clk'event and clk='1';
ahbarbtomst.HADDR<=reg_addr;
ahbarbtomst.HTRANS<="10";
wait until clk'event and clk='1';
ahbarbtomst.HBUSREQ<='0';
ahbarbtomst.HTRANS<="00";
ahbarbtomst.HWDATA<=reg_data;

wait until clk'event and clk='1';
ahbarbtomst.HBUSREQ<='0';

--lire
--verification de la cohérenec des données
if  regdata=ahbmsttoarb.HRDATA then

 print("module single " & hstr(reg_data) & "  OK ");
else
 print("module " & hstr(reg_data) & "  ECHEC ");
end if;

regaddrmul<=regaddr * tmpmul;
regdatamul<=regdata * tmpmul;

wait until clk'event and clk='1';
regaddr<=regaddrmul(31 downto 0);
regdata<=regdatamul(31 downto 0);

wait until clk'event and clk='1';


--burst mode ecriture

ahbarbtomst.HBUSREQ<='1';
ahbarbtomst.HADDR<=regaddr;
ahbarbtomst.HWDATA<= regdata ;

wait until clk'event and clk='1';
ahbarbtomst.HWRITE<='1';
ahbarbtomst.HTRANS<="10";

wait until clk'event and clk='1';
for i in 1 to 10 loop
```

```
regaddr <=regaddr+ 1;
regdata <=regdata +1;
ahbarbtomst.HADDR<=regaddr+ 1;
ahbarbtomst.HWDATA<= regdata + 1;
ahbarbtomst.HWRITE<='1';
ahbarbtomst.HTRANS<="11";
wait until clk'event and clk='1';

end loop;
ahbarbtomst.HTRANS<="00";
ahbarbtomst.HBUSREQ<='0';
ahbarbtomst.HWDATA<= regdata + 1;

wait until clk'event and clk='1';
wait until clk'event and clk='1';
wait until clk'event and clk='1';
regaddr<=regaddrmul(31 downto 0);
regdata<=regdatamul(31 downto 0);

wait until clk'event and clk='1';
--burst mode lecture

ahbarbtomst.HBUSREQ<='1';
ahbarbtomst.HADDR<=regaddr;
ahbarbtomst.HWDATA<= regdata ;

wait until clk'event and clk='1';
ahbarbtomst.HWRITE<='0';
ahbarbtomst.HTRANS<="10";

wait until clk'event and clk='1';
for i in 1 to 10 loop
regaddr <=regaddr+ 1;
regdata <=regdata +1;
ahbarbtomst.HADDR<=regaddr+ 1;
ahbarbtomst.HWDATA<= regdata + 1;
ahbarbtomst.HWRITE<='0';
ahbarbtomst.HTRANS<="11";

wait until clk'event and clk='1';

if  regdata=ahbmsttoarb.HRDATA then
--print("module burst " & hstr(reg_data) &  "   " & hstr(regdata) &"  OK
");
else
print("module burst " & hstr(reg_data) & "   " &
hstr(ahbmsttoarb.HRDATA) & "   " & "  ECHEC ");
end if;

end loop;
ahbarbtomst.HTRANS<="00";
ahbarbtomst.HBUSREQ<='0';
regdata <=regdata +1;
```

```
wait until clk'event and clk='1';

if  regdata=ahbmsttoarb.HRDATA then
--print("module burst " & hstr(reg_data) &  "  " & hstr(regdata) &"  OK
");
else
print("module burst" & hstr(reg_data) & "  " & hstr(ahbmsttoarb.HRDATA)
& "  ECHEC ");
end if;


print( "  Fin ");

wait;
END PROCESS;

END AHBMaster1;
```

```
--
================================================================================
-------
-- Design unit   : AMBA (Package declaration)
--
-- File name     : amba.vhd
--
-- Purpose       : This package declares types to be used with the
--                 Advanced Microcontroller Bus Architecture (AMBA).
--
-- Reference     : AMBA(TM) Specification (Rev 2.0), ARM IHI 0011A,
--                 13th May 1999, issue A, first release, ARM Limited
--
--                 The document can be retrieved from http://www.arm.com
--
--                 AMBA is a trademark of ARM Limited.

--                 ARM is a registered trademark of ARM Limited.
--
-- Note          : Naming convention according to AMBA(TM)
Specification:
--                    Signal names are in upper case, except for the
following:
--                       A lower case 'n' in the name indicates that the
signal
--                       is active low.
--                    Constant names are in upper case.
--
--                 The least significant bit of an array is located to
the right,
--                 carrying the index number zero.
--

-- Library       : AMBA_Lib {recommended}
--
-- Author        : European Space Agency (ESA)
--                 P.O. Box 299
--                 NL-2200 AG Noordwijk ZH
--                 The Netherlands
--
-- Contact       : mailto:microelectronics@estec.esa.int
--                 http://www.estec.esa.int/microelectronics
--
-- Copyright (C): European Space Agency (ESA) 2000.
--                 This source code is free software; you can
redistribute it
--                 and/or modify it under the terms of the GNU Lesser
General
--                 Public License as published by the Free Software
Foundation;
--                 either version 2 of the License, or (at your option)
any
```

```
--                 later version. For full details of the license see
file
--
http://www.estec.esa.int/microelectronics/core/copying.lgpl
--
--                 It is recommended that any use of this VHDL source
code is
--                 reported to the European Space Agency. It is also
recommended
--                 that any use of the VHDL source code properly
acknowledges the
--                 European Space Agency as originator.
--
-- Disclaimer    : All information is provided "as is", there is no
warranty that
--                 the information is correct or suitable for any
purpose,
--                 neither implicit nor explicit. This information does
not
--                 necessarily reflect the policy of the European Space
Agency.
-----------------------------------------------------------------------
---------
-- Version  Author   Date           Changes
--
-- 0.2      ESA       5 Jul 2000    Package created
-- 0.3      ESA      10 Jul 2000    Additional HREADY slave input,
--                                  Std_ULogic usage for non-array
signals,
--                                  Additional comments on casing and
addressing
-- 0.4      ESA      14 Jul 2000    HRESETn removed from AHB Slave
input record
--                                  Additional comments on clocking and
reset
--                                  Additional comments on AHB
endianness
--                                  Additional comments on APB
addressing
-- 0.5      ESA      30 Aug 2000    Vector types for AHB
arbiter/decoder and

--                                    APB bridge refined and
corresponding
--                                    record types removed
--                                  Name suffix 'x' removed
--          ESA      04 Feb 2002    Changed copyright text
-----------------------------------------------------------------------
---------

library IEEE;
use IEEE.Std_Logic_1164.all;

package AMBA is
```

```
   ------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) Advanced High-performance Bus (AHB)
   ------------------------------------------------------------------
---------
   -- Records are defined for the input and output of an AHB Master, as
well as
   -- for an AHB Slave. These records are grouped in arrays, for
scalability,
   -- and new records using these arrays are defined for the input and
output of
   -- an AHB Arbiter/Decoder.
   --
   -- The routing of the clock and reset signals defined in the
AMBA(TM)
   -- Specification is not covered in this package, since being
dependent on
   -- the clock and reset conventions defined at system level.
   --
   -- The HCLK and HRESETn signals are routed separately:
   --    HCLK:       Std_ULogic;                          -- rising edge
   --    HRESETn:    Std_ULogic;                          -- active low
reset
   --
   -- The address bus HADDR contains byte addresses. The relation
between the
   -- byte address and the n-byte data bus HDATA can either be little-
endian or
   -- big-endian according to the AMBA(TM) Specification.
   --
   -- It is recommended that only big-endian modules are implemented
using
   -- this package.
   --
   ------------------------------------------------------------------
---------
   -- Constant definitions for AMBA(TM) AHB
   ------------------------------------------------------------------
---------
   constant HDMAX:   Positive range 32 to 1024 := 32;    -- data width
   constant HAMAX:   Positive range 32 to 32   := 32;    -- address
width

   ------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) AHB Masters
   ------------------------------------------------------------------
---------
   -- AHB master inputs (HCLK and HRESETn routed separately)
   type AHB_Mst_In_Type is
      record
         HGRANT:     Std_ULogic;                          -- bus grant
         HREADY:     Std_ULogic;                          -- transfer
done
```

```
        HRESP:      Std_Logic_Vector(1       downto 0); -- response
type
        HRDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- read data
bus
    end record;

  -- AHB master outputs
  type AHB_Mst_Out_Type is
    record
      HBUSREQ:    Std_ULogic;                          -- bus request

      HLOCK:      Std_ULogic;                          -- lock
request
      HTRANS:     Std_Logic_Vector(1       downto 0); -- transfer
type
      HADDR:      Std_Logic_Vector(HAMAX-1 downto 0); -- address bus
(byte)
      HWRITE:     Std_ULogic;                          -- read/write
      HSIZE:      Std_Logic_Vector(2       downto 0); -- transfer
size
      HBURST:     Std_Logic_Vector(2       downto 0); -- burst type
      HPROT:      Std_Logic_Vector(3       downto 0); -- protection
control
      HWDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- write data
bus
    end record;

  ------------------------------------------------------------------
---------
  -- Definitions for AMBA(TM) AHB Slaves
  ------------------------------------------------------------------
---------
  -- AHB slave inputs (HCLK and HRESETn routed separately)
  type AHB_Slv_In_Type is
    record
      HSEL:       Std_ULogic;                          -- slave
select
      HADDR:      Std_Logic_Vector(HAMAX-1 downto 0); -- address bus
(byte)
      HWRITE:     Std_ULogic;                          -- read/write
      HTRANS:     Std_Logic_Vector(1       downto 0); -- transfer
type
      HSIZE:      Std_Logic_Vector(2       downto 0); -- transfer
size
      HBURST:     Std_Logic_Vector(2       downto 0); -- burst type
      HWDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- write data
bus
      HPROT:      Std_Logic_Vector(3       downto 0); -- protection
control
      HREADY:     Std_ULogic;                          -- transfer
done
      HMASTER:    Std_Logic_Vector(3       downto 0); -- current
master
```

```
        HMASTLOCK:  Std_ULogic;                                -- locked
access
      end record;

   -- AHB slave outputs
   type AHB_Slv_Out_Type is
      record
         HREADY:     Std_ULogic;                               -- transfer
done
         HRESP:      Std_Logic_Vector(1       downto 0); -- response
type
         HRDATA:     Std_Logic_Vector(HDMAX-1 downto 0); -- read data
bus
         HSPLIT:     Std_Logic_Vector(15      downto 0); -- split
completion
      end record;

   ------------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) AHB Arbiter/Decoder
   ------------------------------------------------------------------------
---------
   -- supporting array types
   type AHB_Mst_In_Vector  is array (Natural range <> ) of
AHB_Mst_In_Type;
   type AHB_Mst_Out_Vector is array (Natural range <> ) of
AHB_Mst_Out_Type;
   type AHB_Slv_In_Vector  is array (Natural range <> ) of
AHB_Slv_In_Type;
   type AHB_Slv_Out_Vector is array (Natural range <> ) of
AHB_Slv_Out_Type;

   ------------------------------------------------------------------------
---------
   -- Auxiliary constant definitions for AMBA(TM) AHB
   ------------------------------------------------------------------------
---------
   -- constants for HTRANS (transition type, slave output)
   constant HTRANS_IDLE:   Std_Logic_Vector(1 downto 0) := "00";
   constant HTRANS_BUSY:   Std_Logic_Vector(1 downto 0) := "01";
   constant HTRANS_NONSEQ: Std_Logic_Vector(1 downto 0) := "10";
   constant HTRANS_SEQ:    Std_Logic_Vector(1 downto 0) := "11";

   -- constants for HBURST (burst type, master output)
   constant HBURST_SINGLE: Std_Logic_Vector(2 downto 0) := "000";
   constant HBURST_INCR:   Std_Logic_Vector(2 downto 0) := "001";
   constant HBURST_WRAP4:  Std_Logic_Vector(2 downto 0) := "010";
   constant HBURST_INCR4:  Std_Logic_Vector(2 downto 0) := "011";
   constant HBURST_WRAP8:  Std_Logic_Vector(2 downto 0) := "100";
   constant HBURST_INCR8:  Std_Logic_Vector(2 downto 0) := "101";
   constant HBURST_WRAP16: Std_Logic_Vector(2 downto 0) := "110";
   constant HBURST_INCR16: Std_Logic_Vector(2 downto 0) := "111";

   -- constants for HSIZE (transfer size, master output)
```

```
   constant HSIZE_BYTE:    Std_Logic_Vector(2 downto 0) := "000";
   constant HSIZE_HWORD:   Std_Logic_Vector(2 downto 0) := "001";
   constant HSIZE_WORD:    Std_Logic_Vector(2 downto 0) := "010";
   constant HSIZE_DWORD:   Std_Logic_Vector(2 downto 0) := "011";
   constant HSIZE_4WORD:   Std_Logic_Vector(2 downto 0) := "100";
   constant HSIZE_8WORD:   Std_Logic_Vector(2 downto 0) := "101";
   constant HSIZE_16WORD:  Std_Logic_Vector(2 downto 0) := "110";
   constant HSIZE_32WORD:  Std_Logic_Vector(2 downto 0) := "111";

   -- constants for HRESP (response, slave output)
   constant HRESP_OKAY:    Std_Logic_Vector(1 downto 0) := "00";
   constant HRESP_ERROR:   Std_Logic_Vector(1 downto 0) := "01";
   constant HRESP_RETRY:   Std_Logic_Vector(1 downto 0) := "10";
   constant HRESP_SPLIT:   Std_Logic_Vector(1 downto 0) := "11";

   ------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) Advanced Peripheral Bus (APB)
   ------------------------------------------------------------------
---------
   -- Records are defined for the input and output of an APB Slave.
These
   -- records are grouped in arrays, for scalability, and new records
using
   -- these arrays are defined for the input and output of an APB
Bridge.
   --
   -- The routing of the clock and reset signals defined in the
AMBA(TM)
   -- Specification is not covered in this package, since being
dependent on
   -- the clock and reset conventions defined at system level.
   --
   -- The PCLK and PRESETn signals are routed separately:
   --    PCLK:       Std_ULogic;                          -- rising edge
   --    PRESETn:    Std_ULogic;                          -- active low
reset
   --
   -- The characteristics of the address bus PADDR are undefined in the
   -- AMBA(TM) Specification.
   --
   -- When implementing modules with this package, it is recommended
that the
   -- information on the address bus PADDR is interpreted as byte
addresses, but
   -- it should only be used for 32-bit word addressing, i.e. the value
of
   -- address bits 0 and 1 should always be logical 0. For modules not
   -- supporting full 32-bit words on the data bus PDATA, e.g. only
supporting
   -- 16-bit halfwords or 8-bit bytes, the addressing will still be
word based.
   -- Consequently, one halfword or byte will be accessed for each word
address.
```

```
   -- Modules only supporting byte sized data should exchange data on
bit 7 to 0
   -- on the PDATA data bus. Modules only supporting halfword sized
data should
   -- exchange data on bit 15 to 0 on the PDATA data bus. Modules
supporting
   -- word sized data should exchange data on bit 31 to 0 on the PDATA
data bus.
   --
   ------------------------------------------------------------------
---------
   -- Constant definitions for AMBA(TM) APB
   ------------------------------------------------------------------
---------
   constant PDMAX:   Positive range 8 to 32 := 32;       -- data width
   constant PAMAX:   Positive range 8 to 32 := 32;       -- address
width

   ------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) APB Slaves
   ------------------------------------------------------------------
---------
   -- APB slave inputs (PCLK and PRESETn routed separately)
   type APB_Slv_In_Type is
      record
         PSEL:       Std_ULogic;                          -- slave
select

         PENABLE:    Std_ULogic;                          -- strobe
         PADDR:      Std_Logic_Vector(PAMAX-1 downto 0); -- address bus
(byte)
         PWRITE:     Std_ULogic;                          -- write
         PWDATA:     Std_Logic_Vector(PDMAX-1 downto 0); -- write data
bus
      end record;

   -- APB slave outputs
   type APB_Slv_Out_Type is
      record
         PRDATA:     Std_Logic_Vector(PDMAX-1 downto 0); -- read data
bus
      end record;

   ------------------------------------------------------------------
---------
   -- Definitions for AMBA(TM) APB Bridge
   ------------------------------------------------------------------
---------
   -- supporting array types
   type APB_Slv_In_Vector  is array (Natural range <> ) of
APB_Slv_In_Type;
   type APB_Slv_Out_Vector is array (Natural range <> ) of
APB_Slv_Out_Type;
```

```
end AMBA; --
=============================================================================--
```

```
-------------------------------------------------------------------------
--------
-- Titre         : Arbitre par encodeur de priorité
-- Projet        : Bus générique
-------------------------------------------------------------------------
--------
-- Fichier               : arbiter_arbiter.vhd
-- Auteur(s)             : Mathieu Dubois
-- Création              : 4/8/2003
-- Dernière modification : 4/8/2003
-------------------------------------------------------------------------
--------
-- Description   : Arbitre du système
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois.La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Matih
uDubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE IEEE.Std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.ALL;


ENTITY arbiter IS
   GENERIC(
      masters : integer := 8
   );
   PORT(
      hresetn     : IN     Std_ULogic;
      hclk        : IN     Std_ULogic;
      hbusreq     : IN     unsigned (MASTERS-1 DOWNTO 0);
      hgrant      : OUT    unsigned (MASTERS-1 DOWNTO 0);
      haddr_a     : IN     std_logic_vector (31 DOWNTO 0);
      htrans_a    : IN     std_logic_vector (1 DOWNTO 0);
      hburst_a    : IN     std_logic_vector (2 DOWNTO 0);
      hsplit_a    : IN     std_logic_vector (15 DOWNTO 0);
      hresp_a     : IN     std_logic_vector (1 DOWNTO 0);
      hready_a    : IN     std_logic;
      hmaster_a   : OUT    unsigned (3 DOWNTO 0);
      hmastlock_a : OUT    std_logic;
```

```
      hlock       : IN     unsigned (MASTERS-1 DOWNTO 0)
   );

-- Declarations

END arbiter ;

-- hds interface_end
ARCHITECTURE arbiter OF arbiter IS

signal hgrant_int: unsigned (4 DOWNTO 0);
signal hmaster_a_int,hmaster_a_int2,compteur: unsigned (3 DOWNTO 0);
signal hbusreqgen,hgrant_intgen : unsigned (15 DOWNTO 0);

BEGIN

hmastlock_a <= '0';
hbusreqgen(masters-1 DOWNTO 0)<=hbusreq;
hbusreqgen(15 DOWNTO MASTERS)<=(OTHERS=>'0');

--encodeur de priorité

hgrant_intgen<= "0000000000000001" when hbusreqgen(0) = '1' else
                "0000000000000010" when hbusreqgen(1) = '1' else
                "0000000000000100" when hbusreqgen(2) = '1' else
                "0000000000001000" when hbusreqgen(3) = '1' else

                "0000000000010000" when hbusreqgen(4) = '1' else
                "0000000000100000" when hbusreqgen(5) = '1' else
                "0000000001000000" when hbusreqgen(6) = '1' else
                "0000000010000000" when hbusreqgen(7) = '1' else
                "0000000100000000" when hbusreqgen(8) = '1' else
                "0000001000000000" when hbusreqgen(9) = '1' else
                "0000010000000000" when hbusreqgen(10) = '1' else
                "0000100000000000" when hbusreqgen(11) = '1' else
                "0001000000000000" when hbusreqgen(12) = '1' else
                "0010000000000000" when hbusreqgen(13) = '1' else
                "0100000000000000" when hbusreqgen(14) = '1' else
                "1000000000000000" when hbusreqgen(15) = '1' else
                "0000000000000000";

--choix des maitres

master :process(hgrant_intgen)
  begin
case   hgrant_intgen is

         when "0000000000000001"=> hmaster_a_int<= "0000";
         when "0000000000000010"=> hmaster_a_int<= "0001";
         when "0000000000000100"=> hmaster_a_int<= "0010";
         when "0000000000001000"=> hmaster_a_int<= "0011";
         when "0000000000010000"=> hmaster_a_int<= "0100";
         when "0000000000100000"=> hmaster_a_int<= "0101";
         when "0000000001000000"=> hmaster_a_int<= "0110";
```

```
          when "0000000010000000"=> hmaster_a_int<= "0111";
          when "0000000100000000"=> hmaster_a_int<= "1000";
          when "0000001000000000"=> hmaster_a_int<= "1001";
          when "0000010000000000"=> hmaster_a_int<= "1010";
          when "0000100000000000"=> hmaster_a_int<= "1011";
          when "0001000000000000"=> hmaster_a_int<= "1100";
          when "0010000000000000"=> hmaster_a_int<= "1101";
          when "0100000000000000"=> hmaster_a_int<= "1110";
          when "1000000000000000"=> hmaster_a_int<= "1111";

          when OTHERS=> hmaster_a_int<= "0000";
          end case;
  end process;


--synchronise avec l'horloge

encodeur :process(hclk, hresetn)
  begin

   if hresetn = '0' then
        hgrant<= (others=>'0');
        hmaster_a<= (others=>'0');
        hmaster_a_int2<= (others=>'0');
     elsif hclk'event and hclk = '1' then
        hgrant<=hgrant_intgen(MASTERS-1 DOWNTO 0);
        hmaster_a_int2<=hmaster_a_int;
        hmaster_a<=hmaster_a_int2;
    end if;
  end process;

end arbiter;
```

```
-------------------------------------------------------------------------
--------
-- Titre         : Générer l'horloge
-- Projet        : Bus generique
-------------------------------------------------------------------------
--------
-- Fichier               : clocker_behav.vhd (1)
-- Auteur(s)             : Mathieu Dubois
-- Cr?ation              : 15/3/2003
-- Derni?re modification : 22/5/2003
-------------------------------------------------------------------------
--------
-- Description   : Générateur d'horloge
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriete proprietaire de Mathieu Dubois. La possession,les
modifications ou
-- l'utilisation de ce fichier requiert une license ecrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
-- 5/6/2003  Mame Maria MBAYE
-------------------------------------------------------------------------
--------
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY clocker IS
   PORT(
      clk     : OUT    std_logic;
      clkl    : OUT    std_logic;
      clkv    : OUT    std_logic;
      reset_n : OUT    std_logic
   );

-- Declarations

END clocker ;

architecture behav of clocker is

    SIGNAL clkl_d : std_logic := '0';
    SIGNAL clkv_d : std_logic := '0';
    SIGNAL clk_d : std_logic := '0';

BEGIN
    reset : PROCESS
    BEGIN
        reset_n <= '0', '1' after 170 ns;
        WAIT;
```

```
    END PROCESS;

    clkl<= clkl_d;  --clock des modules
    clk<=clk_d;     --clock du systeme
    clkv<= clkv_d;  --clock des modules
```

```
        PROCESS
              VARIABLE i  : integer := 0;
          BEGIN  -- PROCESS
              IF (i = 10000000) THEN
                  i := 0;
              END IF;
        --Diviseur d'horloge par 8
              IF ((i MOD 8)  = 0) THEN
                  clkl_d <= NOT clkl_d;
              END IF;
      --Diviseur d'horloge par 4
              IF ((i MOD 4)  = 0) THEN
                  clkv_d <= NOT clkv_d;
              END IF;

              clk_d <= NOT clk_d;
              i := i+1;
              WAIT FOR  3125 ps;

        END PROCESS;
end behav;
```

```
--------------------------------------------------------------------------
--------
-- Titre          : Décodeur d'adresse
-- Projet         : Bus générique
--------------------------------------------------------------------------
--------
-- Fichier                : decodeur_decodeur.vhd
-- Auteur(s)              : Mathieu Dubois
-- Cr?ation               : 22/8/2003
-- Derni?re modification  : 22/8/2003
--------------------------------------------------------------------------
--------
-- Description    : Décodeur d'adresse
--------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriete proprietaire de Mathieu Dubois. La possession,les
modifications ou
-- l'utilisation de ce fichier requiert une license ecrite de Mathieu
Dubois.
--------------------------------------------------------------------------
--------
-- L'historique de modification :
--
--------------------------------------------------------------------------
--------

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE IEEE.Std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

LIBRARY dynamic;
USE dynamic.AMBA.ALL;

ENTITY decodeur IS
   GENERIC(
      slaves : integer := 16
   );
   PORT(
      haddr_d : IN     std_logic_vector (31 DOWNTO 0);
      hsel_d  : OUT    unsigned (SLAVES-1 DOWNTO 0)
   );

-- Declarations

END decodeur ;

-- hds interface_end
ARCHITECTURE decodeur OF decodeur IS

SIGNAL tmp :std_logic_vector (SLAVES-1 DOWNTO 0);
```

```
BEGIN
--selection de l'esclave a partir de l'adresse
 tmp<=(OTHERS=>'0');
 hsel_d(SLAVES-1 DOWNTO 0)<= unsigned(tmp);
END decodeur;
```

```
-------------------------------------------------------------------------
--------
-- Titre         : Package interne home
-- Projet        : Bus générique
-------------------------------------------------------------------------
--------
-- Fichier               : home_pkg.vhd
-- Auteur(s)             : Mathieu Dubois
-- Creation              : 4/8/2003
-- Derniere modification : 4/8/2003
-------------------------------------------------------------------------
--------
-- Description   : Declaration des registres vecteurs
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriete proprietaire de Mathieu Dubois. La possession,les
modifications ou
-- l'utilisation de ce fichier requiert une license ecrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------

library IEEE;
use IEEE.Std_Logic_1164.all;

package home is
type registre_Vector is array (NATURAL range <>) of std_logic_vector(31
DOWNTO 0);

end home;
```

```
-------------------------------------------------------------------------
--------
-- Titre         : Matrix d'interconnexion
-- Projet        : Bus generique
-------------------------------------------------------------------------
--------
-- Fichier               : matrix_interconnect.vhd
-- Auteur(s)             : Mathieu Dubois
-- Creation              : 22/8/2003
-- Derniere modification : 22/8/2003
-------------------------------------------------------------------------
--------
-- Description   : Décodeur d'adresse
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriete proprietaire de Mathieu Dubois. La possession,les
modifications ou
-- l'utilisation de ce fichier requiert une license ecrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.all;
USE IEEE.Std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;

ENTITY matrix IS
   GENERIC(
      masters : integer := 16;
      SLAVES  : integer := 16
   );
   --  number of masters
   --  number of slaves
   --
   PORT(
      ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
      ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
      hclk        : IN     Std_ULogic;
      hresetn     : IN     Std_ULogic;
      hlock       : OUT    unsigned (MASTERS-1 DOWNTO 0);
      hbusreq     : OUT    unsigned (MASTERS-1 DOWNTO 0);
      haddr_d     : OUT    std_logic_vector (31 DOWNTO 0);
      haddr_a     : OUT    std_logic_vector (31 DOWNTO 0);
      htrans_a    : OUT    std_logic_vector (1 DOWNTO 0);
```

```
      hburst_a     : OUT    std_logic_vector (2 DOWNTO 0);
      hsplit_a     : OUT    std_logic_vector (15 DOWNTO 0);
      hresp_a      : OUT    std_logic_vector (1 DOWNTO 0);
      hready_a     : OUT    std_logic;
      hmaster_a    : IN     unsigned (3 DOWNTO 0);
      hmastlock_a  : IN     std_logic;
      hsel_d       : IN     unsigned (SLAVES-1 DOWNTO 0);
      ahbarbtoslv  : OUT    ahb_slv_in_vector (0 TO SLAVES-1);
      ahbarbtomst  : OUT    ahb_mst_in_vector (0 TO MASTERS-1);

      hgrant       : IN     unsigned (MASTERS-1 DOWNTO 0)
   );

-- Declarations

END matrix ;

-- hds interface_end


-- hds interface_end
ARCHITECTURE interconnect OF matrix IS
--signaux interne AMBA
type reg_type is record
--vhmaster : integer range 0 to MASTERS -1;
vhmaster : integer range 0 to MASTERS -1;
vhmasterd : integer range 0 to MASTERS -1;
slaves : integer range 0 to SLAVES -1;
end record;

  signal r          : reg_type;
  signal haddr      : std_logic_vector(31 downto 0);
  signal hwrite     : std_logic;
  signal hsize      : std_logic_vector(2 downto 0);
  signal hburst     : std_logic_vector(2 downto 0);
  signal htrans     : std_logic_vector(1 downto 0);
  signal hwdata     : std_logic_vector(31 downto 0);
  signal hrdata     : std_logic_vector(31 downto 0);
  signal hresp      : std_logic_vector(1 downto 0);
  signal hready     : std_logic;
  signal hsplit     : std_logic_vector(15 downto 0);
  signal hmaster    : unsigned (3 downto 0);
  signal hmaster_del1    : unsigned (3 downto 0);
  signal hmasterlock : std_logic;
  signal hsel_del_d    : unsigned (SLAVES-1 DOWNTO 0);

 -- signal hsel_del0_d0 : std_logic;
BEGIN


  -- Route remaining Arbiter and Decoder signals --
  haddr_d     <= haddr;
  haddr_a     <= haddr;
  htrans_a    <= htrans;
```

```
  hburst_a    <= hburst;
  hresp_a     <= hresp;
  hready_a    <= hready;
  hsplit_a    <= hsplit;
  hmaster     <= hmaster_a;
  hmasterlock  <= hmastlock_a;
```

```
comb:
process(haddr,htrans,hwrite,hsize,hburst,hready,hwdata,
        hsel_d,hmaster_a,hmastlock_a,hgrant,hrdata,hresp,ahbmsttoarb)
begin

--Route input signals masters
for i in 0 to (masters -1) loop
ahbarbtomst(i).hgrant  <= hgrant(i);
ahbarbtomst(i).hready  <= hready;
ahbarbtomst(i).hrdata  <= hrdata;
ahbarbtomst(i).hresp   <= hresp;

end loop;

for i in 0 to (masters -1) loop
hbusreq(i)<=ahbmsttoarb(i).hbusreq;
hlock(i)<=ahbmsttoarb(i).hlock;
end loop;

--Route input slave
for i in 0 to (slaves -1) loop
ahbarbtoslv(i).haddr<=haddr;
ahbarbtoslv(i).htrans<=htrans;
ahbarbtoslv(i).hwrite<=hwrite;
ahbarbtoslv(i).hsize<=hsize;
ahbarbtoslv(i).hburst<=hburst;
ahbarbtoslv(i).hready<=hready;
ahbarbtoslv(i).hwdata<=hwdata;

ahbarbtoslv(i).hsel<='1';

ahbarbtoslv(i).hmaster<=CONV_STD_LOGIC_VECTOR(hmaster_a,4);
ahbarbtoslv(i).hmastlock<=hmastlock_a;
end loop;

end process;

-- Generate delayed HMASTER
  synch_hmaster: process(hclk, hresetn)
```

```
  begin
    if hresetn = '0' then
      hmaster_del1 <= "0000";
    elsif hclk'event and hclk = '1' then
      hmaster_del1 <= hmaster;
    end if;
  end process;

-- Generate delayed HSEL
  synch_hsel: process(hclk, hresetn)
  begin
    if hresetn = '0' then
       for i in 0 to (slaves -1) loop
           hsel_del_d(i)<='0';
       end loop;
    elsif hclk'event and hclk = '1' then

       if hready = '1' then
       for i in 0 to (slaves -1) loop
           hsel_del_d(i)<=hsel_d(i);
       end loop;
       end if;

    end if;
  end process;

  -- Select proper inputs for master mux --
  mastermux:process(hmaster,hmaster_del1,ahbmsttoarb)
  variable rv : reg_type;
  begin

    rv.vhmaster:=conv_integer(hmaster);
    rv.vhmasterd:=conv_integer(hmaster_del1);
    haddr  <= ahbmsttoarb(rv.vhmaster).haddr;

    hwrite <= ahbmsttoarb(rv.vhmaster).hwrite ;
    hsize  <= ahbmsttoarb(rv.vhmaster).hsize ;
    hburst <= ahbmsttoarb(rv.vhmaster).hburst;
    htrans <= ahbmsttoarb(rv.vhmaster).htrans;
    hwdata <= ahbmsttoarb(rv.vhmasterd).hwdata;
  end process;

  -- Select proper inputs for slave mux --
  slavemux: process(hsel_del_d, ahbslvtoarb)
  variable r : reg_type;
  begin
    r.slaves:=conv_integer(hsel_del_d);
    hready  <= ahbslvtoarb(r.slaves).hready ;
    hresp <= ahbslvtoarb(r.slaves).hresp;
    hrdata <= ahbslvtoarb(r.slaves).hrdata;
  end process;
```

```
END interconnect;
```

```
-- hds header_start
--
-- VHDL Entity dynamic.mem_1r_1w.symbol
--
-- Created:
--          by - ududbois.etudiant (papineau)
--          at - 22:50:56 08/06/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
-- hds header_end
-- ------------------------------------------------------------------------
---------
--  Title       : Memory Manager - Address Counter
--  Project     : Protocol Conversion
-- ------------------------------------------------------------------------
---------
--  File        : memory_3r_1w.vhdl
--  Author      : Mame Maria MBAYE
--  Company     : Ecole Polytechnique de Montreal
--  Created     : 2002/02/03
--  Last update : 2002/02/03
--  Platform    : Simulation By Actel ActiveHDL Revision: ???
--                Synthesis By design_vision Version ???
-- ------------------------------------------------------------------------
---------
--  Description : This bock is a memory with one read port and one
write port
--
--  Parameters :
--
--
-- ------------------------------------------------------------------------
---------
--  Copyright (c) 2001 Ecole Polytechnique de Montreal.
--  Confidential and Proprietary Information
-- ------------------------------------------------------------------------
---------
--  Revisions  :
--  Date        Version  Author      Description
--  2002/02/03  1.0      mmm        Creation of this file
-- ------------------------------------------------------------------------
---------
-- ------------------------------------------------------------------------
---------
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.std_logic_arith.all;

ENTITY mem_1r_1w IS
   --  length of the address path
```

```
   --
   GENERIC(
      addr_width : INTEGER range 0 to 1000;
      word_width : INTEGER range 0 to 64 := 32;
      mem_depth  : INTEGER range 0 to 20000
   );
   PORT(
      -- The clock and the reset
      sys_clk    : IN     STD_LOGIC;
      -- 3 read ports
      mem_rd_n   : IN     STD_LOGIC;
      read_addr  : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
      -- write port
      mem_wr_n   : IN     STD_LOGIC;
      write_addr : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
      write_word : IN     STD_LOGIC_VECTOR (word_width-1 DOWNTO 0);
      -- the output word depending on input addresses
      read_word  : OUT    STD_LOGIC_VECTOR (word_width-1 DOWNTO 0)
   );

-- Declarations

END mem_1r_1w ;

-- hds interface_end

--------------------------------------------------------------------------
--------
ARCHITECTURE mem_1r_1w_rtl OF mem_1r_1w IS
    TYPE mem_type IS ARRAY (mem_depth-1 DOWNTO 0) OF
STD_LOGIC_VECTOR(word_width-1 DOWNTO 0);
    SIGNAL mem : mem_type;
   -- ATTRIBUTE syn_ramstyle : STRING;
    --ATTRIBUTE syn_ramstyle OF mem : SIGNAL IS "block_ram";
    SIGNAL rd_addr : STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
BEGIN  -- architecture

    PROCESS (sys_clk) --, sys_reset_n

    BEGIN  -- PROCESS
      -- activities triggered by asynchronous reset (active low)
       --rd_addr <= read_addr;
            -- rd_addr <= read_addr;
           --read_word <= mem(conv_integer(rd_addr));
      --IF sys_clk = '1' AND sys_clk'EVENT AND mem_wr_n = '0' THEN
        -- mem(conv_integer(write_addr)) <= write_word;
      --ELSIF (sys_clk = '1' AND sys_clk'EVENT AND mem_wr_n = '1') THEN
       --  read_word <= mem(conv_integer(read_addr));
      --END IF;
   IF sys_clk = '1' AND sys_clk'EVENT THEN

        IF mem_wr_n = '0'  THEN
         mem(conv_integer(write_addr)) <= write_word;
        ELSE
```

```
        read_word <= mem(conv_integer(read_addr));
        END IF;

   END IF;
    END PROCESS;
END  mem_1r_1w_rtl;
```

```
------------------------------------------------------------------------
--------
-- Titre         : Esclave AHB
-- Projet        : Bus générique
------------------------------------------------------------------------
--------
-- Fichier               : ptslv_pont.vhd
-- Auteur(s)             : Mathieu Dubois
-- Création              : 8/4/2003
-- Dernière modification : 8/4/2003
------------------------------------------------------------------------
--------
-- Description   : Pont Esclave AHB pour la mémoire
------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriét propriétaire de Mathieu Dubois.La possession, modifications
ou
-- l'utilisation de ce fichier requiert une license écrite de Mathieu
Dubois.
------------------------------------------------------------------------
--------
-- L'historique de modification :
--
------------------------------------------------------------------------
--------


LIBRARY ieee ;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;


ENTITY ptslv IS
   PORT(
      hclk        : IN     std_logic;
      hreset_n    : IN     std_logic;
      readdata    : IN     std_logic_vector (31 DOWNTO 0);
      clk         : OUT    std_logic;
      rd_n        : OUT    std_logic;
      wr_n        : OUT    std_logic;
      writedata   : OUT    std_logic_vector (31 DOWNTO 0);
      addr        : OUT    std_logic_vector (13 DOWNTO 0);
      ahbslavein  : IN     AHB_Slv_In_Type;
      ahbslaveout : OUT    AHB_Slv_Out_Type
   );

-- Declarations
```

```
END ptslv ;

-- hds interface_end

ARCHITECTURE pont OF ptslv IS
signal htrans_int  : std_logic_vector(1 downto 0);
signal addr_int  : std_logic_vector(13 downto 0);
signal write_int : std_logic;
signal check : std_logic;



BEGIN
  -- Fixed values
  AHBSlaveout.hready <= '1';                -- toujours un cycle
d'acces
  AHBSlaveOut.hresp     <= "00";               -- hresp fixe a "OK":
                                    --   nous assumons que les adresses
sont OK

  -- Connect le bus de donne au module directement
  AHBSlaveOut.hrdata    <= readdata;
  writedata <= AHBSlavein.hwdata;
  addr <= addr_int;
  check<=(htrans_int(1) or htrans_int(0)) and write_int;

  wr_n<=not(check);
  rd_n<=check;

  -- Route l'horloge du timer au module
  clk     <=  hclk;


  -- Latch address and control signals, as per AMBA secification
bascule :process(hclk, hreset_n)
  begin

    if hreset_n = '0' then
      write_int <= '0';
      htrans_int<= "00";
      addr_int<= (others=>'0');
     elsif hclk'event and hclk = '1' then

      write_int<=AHBSlaveIn.hwrite;
      addr_int<=AHBSlaveIn.haddr(13 downto 0);
        htrans_int<=AHBSlaveIn.htrans;

    end if;
  end process;


END pont;
```

```
------------------------------------------------------------------------
--------
-- Titre         : Registres initiaux
-- Projet        : Bus générique
------------------------------------------------------------------------
--------
-- Fichier               : registre_registres.vhd
-- Auteur(s)             : Mathieu Dubois
-- Création              : 4/8/2003
-- Dernière modification : 4/8/2003
------------------------------------------------------------------------
--------
-- Description   : Vecteurs pour l'initialisation des émulateurs
------------------------------------------------------------------------
--------

-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois.La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Mathieu
Dubois.
------------------------------------------------------------------------
--------
-- L'historique de modification :
--
------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY dynamic;
USE dynamic.AMBA.ALL;
USE ieee.std_logic_unsigned.ALL;
USE dynamic.home.all;

ENTITY registre IS
   GENERIC(
      masters : integer := 8
   );
   PORT(
      reg_addr : OUT    registre_vector (0 TO MASTERS-1);
      reg_data : OUT    registre_vector (0 TO MASTERS-1)
   );

-- Declarations

END registre ;

ARCHITECTURE registres OF registre IS
signal data : std_logic_vector(31 downto
0):="00000000000000000000000000000001";
```

```
signal addr: std_logic_vector(31 downto
0):="00000000000000000000000000000010";

BEGIN
--Creer un vecteur linéaire pour les signaux d'adresse et de données
init: process
begin
for i in 0 to (masters -1) loop
reg_addr(i)<=addr+i;
reg_data(i)<=data+i;
end loop;
wait;
end process;
END registres;
```

```
-------------------------------------------------------------------------
--------
-- Titre         : Top
-- Projet        : Bus générique
-------------------------------------------------------------------------
--------
-- Fichier               : top_struct.vhd (1)
-- Auteur(s)             : Mathieu Dubois
-- Création              : 6/10/2003
-- Dernière modification : 6/10/2003
-------------------------------------------------------------------------
--------
-- Description   : Top du système
-------------------------------------------------------------------------
--------
-- Copyright (c) 2003 par Mathieu Dubois. Ce model est confidentiel et
une
-- propriété propriétaire de Mathieu Dubois.La possession,
modifications ou
-- l'utilisation de ce fichier requiert une license écrite de Mathieu
Dubois.
-------------------------------------------------------------------------
--------
-- L'historique de modification :
--
-------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;USE dynamic.home.all;
USE ieee.std_logic_unsigned.ALL;

ENTITY top IS
   GENERIC(
      MASTERS : integer := 6;
      SLAVES  : integer := 1
   );
-- Declarations

END top ;

ARCHITECTURE struct OF top IS

   -- Architecture declarations

   -- Internal signal declarations
   SIGNAL addr         : std_logic_vector(13 DOWNTO 0);
   SIGNAL ahbarbtomst  : ahb_mst_in_vector(0 TO MASTERS-1);
   SIGNAL ahbarbtomst1 : ahb_mst_out_vector(0 TO Masters-1);
   SIGNAL ahbarbtoslv  : ahb_slv_in_vector(0 TO SLAVES-1);
```

```
SIGNAL ahbmsttoarb  : ahb_mst_out_vector(0 TO MASTERS-1);
SIGNAL ahbmsttoarb1 : ahb_mst_in_vector(0 TO Masters-1);
SIGNAL ahbslvtoarb  : ahb_slv_out_vector(0 TO SLAVES-1);
SIGNAL clk          : std_logic;
SIGNAL clk1         : std_logic;
SIGNAL clkl         : std_logic;
SIGNAL clkv         : std_logic;
SIGNAL rd_n         : std_logic;
SIGNAL read_word    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL reg_addr     : registre_vector(0 TO MASTERS-1);
SIGNAL reg_data     : registre_vector(0 TO MASTERS-1);
SIGNAL reset_n      : std_logic;
SIGNAL wr_n         : std_logic;
SIGNAL write_word   : STD_LOGIC_VECTOR(31 DOWNTO 0);


-- Component Declarations
COMPONENT AHBArbiterDecoder
GENERIC (
   MASTERS : integer := 16;
   SLAVES  : integer := 16
);
PORT (
   hclk        : IN     Std_ULogic ;
   hresetn     : IN     Std_ULogic ;
   ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
   ahbarbtomst : OUT    ahb_mst_in_vector (0 TO MASTERS-1);
   ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
   ahbarbtoslv : OUT    ahb_slv_in_vector (0 TO SLAVES-1)
);
END COMPONENT;
COMPONENT AHBMaster1
PORT (
   ahbmsttoarb : IN     ahb_mst_in_type ;
   clk         : IN     std_logic ;
   reset_n     : IN     std_logic ;
   ahbarbtomst : OUT    ahb_mst_out_type ;
   reg_addr    : IN     std_logic_vector (31 DOWNTO 0);
   reg_data    : IN     std_logic_vector (31 DOWNTO 0)
);

END COMPONENT;
COMPONENT adaptateur
PORT (
   ahbmstin  : IN     ahb_mst_in_type ;
   ahbslvin  : IN     ahb_mst_out_type ;
   clk       : IN     std_logic ;
   reset_n   : IN     std_logic ;
   ahbmstout : OUT    ahb_mst_out_type ;
   ahbslvout : OUT    ahb_mst_in_type ;
   clkbus    : IN     std_logic
);
END COMPONENT;
COMPONENT clocker
```

```
PORT (
   clk      : OUT    std_logic ;
   clkl     : OUT    std_logic ;
   clkv     : OUT    std_logic ;
   reset_n : OUT    std_logic
);
END COMPONENT;
COMPONENT mem_1r_1w
GENERIC (
   addr_width : INTEGER range 0 to 1000;
   word_width : INTEGER range 0 to 64 := 32;
   mem_depth  : INTEGER range 0 to 20000
);
PORT (
   -- The clock and the reset
   sys_clk    : IN     STD_LOGIC ;
   -- 3 read ports
   mem_rd_n   : IN     STD_LOGIC ;
   read_addr  : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
   -- write port
   mem_wr_n   : IN     STD_LOGIC ;
   write_addr : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
   write_word : IN     STD_LOGIC_VECTOR (word_width-1 DOWNTO 0);
   -- the output word depending on input addresses
   read_word  : OUT    STD_LOGIC_VECTOR (word_width-1 DOWNTO 0)
);
END COMPONENT;
COMPONENT ptslv
PORT (
   hclk        : IN     std_logic ;
   hreset_n    : IN     std_logic ;
   readdata    : IN     std_logic_vector (31 DOWNTO 0);
   clk         : OUT    std_logic ;
   rd_n        : OUT    std_logic ;
   wr_n        : OUT    std_logic ;
   writedata   : OUT    std_logic_vector (31 DOWNTO 0);
   addr        : OUT    std_logic_vector (13 DOWNTO 0);
   ahbslavein  : IN     AHB_Slv_In_Type ;
   ahbslaveout : OUT    AHB_Slv_Out_Type
);
END COMPONENT;
COMPONENT registre
GENERIC (
   masters : integer := 8
);
PORT (
   reg_addr : OUT    registre_vector (0 TO MASTERS-1);
   reg_data : OUT    registre_vector (0 TO MASTERS-1)
);
END COMPONENT;

-- Optional embedded configurations
-- pragma synthesis_off
FOR ALL : AHBArbiterDecoder USE ENTITY dynamic.AHBArbiterDecoder;
```

```
   FOR ALL : AHBMaster1 USE ENTITY dynamic.AHBMaster1;
   FOR ALL : adaptateur USE ENTITY dynamic.adaptateur;
   FOR ALL : clocker USE ENTITY dynamic.clocker;
   FOR ALL : mem_1r_1w USE ENTITY dynamic.mem_1r_1w;
   FOR ALL : ptslv USE ENTITY dynamic.ptslv;
   FOR ALL : registre USE ENTITY dynamic.registre;
   -- pragma synthesis_on


BEGIN
   -- Instance port mappings.
   I0 : AHBArbiterDecoder
      GENERIC MAP (
         MASTERS => MASTERS,
         SLAVES  => SLAVES
      )
      PORT MAP (
         hclk        => clk,
         hresetn     => reset_n,
         ahbmsttoarb => ahbmsttoarb,
         ahbarbtomst => ahbarbtomst,
         ahbslvtoarb => ahbslvtoarb,
         ahbarbtoslv => ahbarbtoslv
      );
   I1 : clocker
      PORT MAP (
         clk     => clk,
         clkl    => clkl,
         clkv    => clkv,
         reset_n => reset_n
      );
   I2 : mem_1r_1w
      GENERIC MAP (
         addr_width => 14,
         word_width => 32,
         mem_depth  => 3000
      )
      PORT MAP (
         sys_clk    => clk1,
         mem_rd_n   => rd_n,
         read_addr  => addr,
         mem_wr_n   => wr_n,
         write_addr => addr,
         write_word => write_word,
         read_word  => read_word
      );
   I4 : ptslv
      PORT MAP (
         hclk        => clk,
         hreset_n    => reset_n,
         readdata    => read_word,
         clk         => clk1,
         rd_n        => rd_n,
```

```
         wr_n        => wr_n,
         writedata   => write_word,
         addr        => addr,
         ahbslavein  => ahbarbtoslv(0),
         ahbslaveout => ahbslvtoarb(0)
      );
   I3 : registre
      GENERIC MAP (
         masters => masters
      )
      PORT MAP (
         reg_addr => reg_addr,
         reg_data => reg_data
      );
--Premier domaine d'horloge
   g0: FOR i IN 0 TO 1 GENERATE
      I6 : adaptateur
         PORT MAP (
            ahbmstin  => ahbarbtomst(i),
            ahbslvin  => ahbarbtomst1(i),
            clk       => clkv,
            reset_n   => reset_n,
            ahbmstout => ahbmsttoarb(i),
            ahbslvout => ahbmsttoarb1(i),
            clkbus    => clk
         );
      I7 : AHBMaster1
         PORT MAP (
            ahbmsttoarb => ahbmsttoarb1(i),
            clk         => clkv,
            reset_n     => reset_n,
            ahbarbtomst => ahbarbtomst1(i),
            reg_addr    => reg_addr(i),
            reg_data    => reg_data(i)
         );
   END GENERATE g0;

--deuxième domaine d'horloge
   g1: FOR i IN 2 TO 5 GENERATE
      I8 : AHBMaster1
         PORT MAP (
            ahbmsttoarb => ahbmsttoarb1(i),
            clk         => clkl,
            reset_n     => reset_n,
            ahbarbtomst => ahbarbtomst1(i),
            reg_addr    => reg_addr(i),
            reg_data    => reg_data(i)
         );
      I9 : adaptateur
         PORT MAP (
            ahbmstin  => ahbarbtomst(i),
            ahbslvin  => ahbarbtomst1(i),
            clk       => clkl,
            reset_n   => reset_n,
```

```
            ahbmstout => ahbmsttoarb(i),
            ahbslvout => ahbmsttoarb1(i),
            clkbus    => clk
        );
    END GENERATE g1;

END struct;
```

```
Fichier total.csh
#! /bin/tcsh -f
dc_shell << EOF

/* Script de compilation pour le DCO */
/*2 modules*/

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=2"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=2,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=2,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=2,SLAVES=1"
/*uniquify*/
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus2.out
report_area   > report_area2.out
write -format db -hier -o dynbus2.db

/*3 modules*/

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
```

```
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=3"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=3,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=3,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=3,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus3.out
report_area   > report_area3.out
write -format db -hier -o dynbus3.db

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=4"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
```

```
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=4,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=4,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=4,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus4.out
report_area   > report_area4.out
write -format db -hier -o dynbus4.db

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=5"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=5,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=5,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=5,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus5.out
report_area   > report_area5.out
write -format db -hier -o dynbus5.db
/*Analyse du filtre fir */
```

```
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=6"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=6,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=6,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=6,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus6.out
report_area   > report_area6.out
write -format db -hier -o dynbus6.db
/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}
```

```
/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=7"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=7,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=7,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=7,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus7.out
report_area   > report_area7.out
write -format db -hier -o dynbus7.db

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=8"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=8,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=8,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=8,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
```

```
report_timing > report_bus8.out
report_area   > report_area8.out
write -format db -hier -o dynbus8.db


/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=9"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=9,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=9,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters "MASTERS=9,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus9.out
report_area   > report_area9.out
write -format db -hier -o dynbus9.db
/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
```

```
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=10"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=10,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=10,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=10,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus10.out
report_area   > report_area10.out
write -format db -hier -o dynbus10.db
/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=11"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=11,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=11,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv
```

```
elaborate -library dynamic synthesetop -parameters
"MASTERS=11,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus11.out
report_area   > report_area11.out
write -format db -hier -o dynbus11.db
/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=12"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=12,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=12,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=12,SLAVES=1"

uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus12.out
report_area   > report_area12.out
write -format db -hier -o dynbus12.db

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=13"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
```

```
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=13,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=13,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=13,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus13.out
report_area   > report_area13.out
write -format db -hier -o dynbus13.db

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=14"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=14,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=14,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=14,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus14.out
report_area   > report_area14.out
```

```
write -format db -hier -o dynbus14.db


/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=15"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=15,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=15,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=15,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus15.out
report_area   > report_area15.out
write -format db -hier -o dynbus15.db

/*Analyse du filtre fir */
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/amba_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/home_pkg.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/arbiter_arbiter.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/decodeur_decodeur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ahbarbiterdecoder_rtl.vhd"}
```

```
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/matrix_interconnect.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/adaptateur_adaptateur.vhd"}
analyze -format vhdl -lib dynamic
{"../complet/V3/RenoirLib/hdl/ptslv_pont.vhd"}
analyze -format vhdl -lib dynamic {"./synthesetop_struct.vhd"}

/* Synthese */
elaborate -library dynamic arbiter -update -parameters "MASTERS=16"
elaborate -library dynamic decodeur -parameters "SLAVES=1"
elaborate -library dynamic ahbarbiterdecoder -parameters
"MASTERS=16,SLAVES=1"
elaborate -library dynamic matrix -parameters "MASTERS=16,SLAVES=1"
elaborate -library dynamic adaptateur
elaborate -library dynamic ptslv

elaborate -library dynamic synthesetop -parameters
"MASTERS=16,SLAVES=1"
uniquify
compile -ungroup_all
remove_unconnected_ports -blast_buses find(-hierarchy cell, "*")
create_clock clk -period 1.12
compile -map_effort high
report_timing > report_bus16.out
report_area   > report_area16.out
write -format db -hier -o dynbus16.db

quit
EOF
```

```
-- hds header_start
--
-- VHDL Entity dynamic.synthesetop.symbol
--
-- Created:
--          by - ududbois.etudiant (isthelene)
--          at - 20:50:36 08/19/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;
USE dynamic.home.all;

ENTITY synthesetop IS
   GENERIC(
      MASTERS : integer := 4;
      SLAVES  : integer := 1
   );
   PORT(
      clk        : IN     std_logic;
      clk1       : IN     std_logic;
      read_word  : IN     STD_LOGIC_VECTOR (31 DOWNTO 0);
      reset_n    : IN     std_logic;
      addr       : OUT    std_logic_vector (13 DOWNTO 0);
      clk1       : OUT    std_logic;
      rd_n       : OUT    std_logic;
      wr_n       : OUT    std_logic;
      write_word : OUT    STD_LOGIC_VECTOR (31 DOWNTO 0)
   );

-- Declarations

END synthesetop ;

-- hds interface_end
--
-- VHDL Architecture dynamic.synthesetop.struct
--
-- Created:
--          by - ududbois.etudiant (isthelene)
--          at - 20:50:37 08/19/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

```
LIBRARY dynamic;

ARCHITECTURE struct OF synthesetop IS

   -- Architecture declarations

   -- Internal signal declarations
   SIGNAL ahbarbtomst  : ahb_mst_in_vector(0 TO MASTERS-1);
   SIGNAL ahbarbtomst1 : ahb_mst_out_vector(0 TO Masters-1);
   SIGNAL ahbarbtoslv  : ahb_slv_in_vector(0 TO SLAVES-1);
   SIGNAL ahbmsttoarb  : ahb_mst_out_vector(0 TO MASTERS-1);
   SIGNAL ahbmsttoarb1 : ahb_mst_in_vector(0 TO Masters-1);
   SIGNAL ahbslvtoarb  : ahb_slv_out_vector(0 TO SLAVES-1);


   -- Component Declarations
   COMPONENT AHBArbiterDecoder
   GENERIC (
      MASTERS : Natural := 16;
      SLAVES  : Natural := 16
   );
   PORT (
      hclk        : IN     Std_ULogic ;
      hresetn     : IN     Std_ULogic ;
      ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
      ahbarbtomst : OUT    ahb_mst_in_vector (0 TO MASTERS-1);
      ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
      ahbarbtoslv : OUT    ahb_slv_in_vector (0 TO SLAVES-1)
   );
   END COMPONENT;
   COMPONENT adaptateur
   PORT (
      ahbmstin  : IN     ahb_mst_in_type ;
      ahbslvin  : IN     ahb_mst_out_type ;
      clk       : IN     std_logic ;
      reset_n   : IN     std_logic ;
      ahbmstout : OUT    ahb_mst_out_type ;
      ahbslvout : OUT    ahb_mst_in_type ;
      clkbus    : IN     std_logic
   );
   END COMPONENT;
   COMPONENT ptslv
   PORT (
      hclk        : IN     std_logic ;
      hreset_n    : IN     std_logic ;
      readdata    : IN     std_logic_vector (31 DOWNTO 0);
      clk         : OUT    std_logic ;
      rd_n        : OUT    std_logic ;
      wr_n        : OUT    std_logic ;
      writedata   : OUT    std_logic_vector (31 DOWNTO 0);
      addr        : OUT    std_logic_vector (13 DOWNTO 0);
      ahbslavein  : IN     AHB_Slv_In_Type ;
      ahbslaveout : OUT    AHB_Slv_Out_Type
```

```
   );
   END COMPONENT;

   -- Optional embedded configurations
   -- pragma synthesis_off
   FOR ALL : AHBArbiterDecoder USE ENTITY dynamic.AHBArbiterDecoder;
   FOR ALL : adaptateur USE ENTITY dynamic.adaptateur;
   FOR ALL : ptslv USE ENTITY dynamic.ptslv;
   -- pragma synthesis_on


BEGIN
   -- Instance port mappings.
   I1 : AHBArbiterDecoder
      GENERIC MAP (
         MASTERS => MASTERS,
         SLAVES  => SLAVES
      )
      PORT MAP (
         hclk        => clk,
         hresetn     => reset_n,
         ahbmsttoarb => ahbmsttoarb,
         ahbarbtomst => ahbarbtomst,
         ahbslvtoarb => ahbslvtoarb,
         ahbarbtoslv => ahbarbtoslv
      );
   I5 : ptslv
      PORT MAP (
         hclk        => clk,
         hreset_n    => reset_n,
         readdata    => read_word,
         clk         => clk1,
         rd_n        => rd_n,
         wr_n        => wr_n,
         writedata   => write_word,
         addr        => addr,
         ahbslavein  => ahbarbtoslv(0),
         ahbslaveout => ahbslvtoarb(0)
      );

   g1: FOR i IN 0 TO MASTERS GENERATE
      I7 : adaptateur
         PORT MAP (
            ahbmstin  => ahbarbtomst(i),
            ahbslvin  => ahbarbtomst1(i),
            clk       => clk1,
            reset_n   => reset_n,
            ahbmstout => ahbmsttoarb(i),
            ahbslvout => ahbmsttoarb1(i),
            clkbus    => clk
         );
   END GENERATE g1;

END struct;
```

```
Fichier gen_total.csh :Générateur du top du schema pour verification
des différentes fréquence des modules

cp ../hdl/*.vhd ../autovhd/autohdl
vlib dynamic
vmap dynamic dynamic

sed 's/MASTERS_PROG/8/g' < top_struct.vhd > tmp1_top_struct.vhd
sed 's/CELLULE1/2/g' < tmp1_top_struct.vhd > tmp2_top_struct.vhd
sed 's/CELLULE2/3/g' < tmp2_top_struct.vhd > tmp3_top_struct.vhd
sed 's/CELLULE3/7/g' < tmp3_top_struct.vhd > autohdl/top_struct.vhd
sed 's/CELLULE1/8/g' < clocker_behav.vhd > tmp_clocker_behav.vhd
sed 's/CELLULE2/4/g' < tmp_clocker_behav.vhd >
autohdl/clocker_behav.vhd
./compile.csh

vsim -t 100ps dynamic.top < run.do |cat >>raport.out


sed 's/MASTERS_PROG/8/g' < top_struct.vhd > tmp1_top_struct.vhd
sed 's/CELLULE1/2/g' < tmp1_top_struct.vhd > tmp2_top_struct.vhd
sed 's/CELLULE2/3/g' < tmp2_top_struct.vhd > tmp3_top_struct.vhd
sed 's/CELLULE3/7/g' < tmp3_top_struct.vhd > autohdl/top_struct.vhd
sed 's/CELLULE1/8/g' < clocker_behav.vhd > tmp_clocker_behav.vhd
sed 's/CELLULE2/4/g' < tmp_clocker_behav.vhd >
autohdl/clocker_behav.vhd
./compile.csh

vsim -t 100ps dynamic.top < run.do |cat >>raport.out

sed 's/MASTERS_PROG/8/g' < top_struct.vhd > tmp1_top_struct.vhd
sed 's/CELLULE1/2/g' < tmp1_top_struct.vhd > tmp2_top_struct.vhd
sed 's/CELLULE2/3/g' < tmp2_top_struct.vhd > tmp3_top_struct.vhd
sed 's/CELLULE3/7/g' < tmp3_top_struct.vhd > autohdl/top_struct.vhd
sed 's/CELLULE1/8/g' < clocker_behav.vhd > tmp_clocker_behav.vhd
sed 's/CELLULE2/4/g' < tmp_clocker_behav.vhd >
autohdl/clocker_behav.vhd
./compile.csh

vsim -t 100ps dynamic.top < run.do |cat >>raport.out

sed 's/MASTERS_PROG/8/g' < top_struct.vhd > tmp1_top_struct.vhd
sed 's/CELLULE1/2/g' < tmp1_top_struct.vhd > tmp2_top_struct.vhd
sed 's/CELLULE2/3/g' < tmp2_top_struct.vhd > tmp3_top_struct.vhd
sed 's/CELLULE3/7/g' < tmp3_top_struct.vhd > autohdl/top_struct.vhd
sed 's/CELLULE1/8/g' < clocker_behav.vhd > tmp_clocker_behav.vhd
sed 's/CELLULE2/4/g' < tmp_clocker_behav.vhd >
autohdl/clocker_behav.vhd
./compile.csh

vsim -t 100ps dynamic.top < run.do |cat >>raport.out
```

```
-- Top_struct.vhd  (2)
-- hds header_start
--
-- VHDL Entity dynamic.top.symbol
--
-- Created:
--          by - udubois.etudiant (papineau)
--          at - 10:55:35 09/26/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

LIBRARY dynamic;
USE dynamic.AMBA.all;USE dynamic.home.all;

ENTITY top IS
   GENERIC(
      MASTERS : integer := MASTERS_PROG;
      SLAVES  : integer := 1
   );
-- Declarations

END top ;

-- hds interface_end
--
-- VHDL Architecture dynamic.top.struct
--
-- Created:
--          by - udubois.etudiant (papineau)
--          at - 10:55:36 09/26/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
LIBRARY dynamic;
USE dynamic.AMBA.ALL;
USE ieee.std_logic_unsigned.ALL;

LIBRARY dynamic;

ARCHITECTURE struct OF top IS

   -- Architecture declarations

   -- Internal signal declarations
   SIGNAL addr         : std_logic_vector(13 DOWNTO 0);
   SIGNAL ahbarbtomst  : ahb_mst_in_vector(0 TO MASTERS-1);
```

```
SIGNAL ahbarbtomst1 : ahb_mst_out_vector(0 TO Masters-1);
SIGNAL ahbarbtoslv  : ahb_slv_in_vector(0 TO SLAVES-1);
SIGNAL ahbmsttoarb  : ahb_mst_out_vector(0 TO MASTERS-1);
SIGNAL ahbmsttoarb1 : ahb_mst_in_vector(0 TO Masters-1);
SIGNAL ahbslvtoarb  : ahb_slv_out_vector(0 TO SLAVES-1);
SIGNAL clk          : std_logic;
SIGNAL clk1         : std_logic;
SIGNAL clkl         : std_logic;
SIGNAL clkv         : std_logic;
SIGNAL rd_n         : std_logic;
SIGNAL read_word    : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL reg_addr     : registre_vector(0 TO MASTERS-1);
SIGNAL reg_data     : registre_vector(0 TO MASTERS-1);
SIGNAL reset_n      : std_logic;
SIGNAL wr_n         : std_logic;
SIGNAL write_word   : STD_LOGIC_VECTOR(31 DOWNTO 0);


-- Component Declarations
COMPONENT AHBArbiterDecoder
GENERIC (
   MASTERS : integer := 16;
   SLAVES  : integer := 16
);
PORT (
   hclk        : IN     Std_ULogic ;
   hresetn     : IN     Std_ULogic ;
   ahbmsttoarb : IN     ahb_mst_out_vector (0 TO MASTERS-1);
   ahbarbtomst : OUT    ahb_mst_in_vector (0 TO MASTERS-1);
   ahbslvtoarb : IN     ahb_slv_out_vector (0 TO SLAVES-1);
   ahbarbtoslv : OUT    ahb_slv_in_vector (0 TO SLAVES-1)
);
END COMPONENT;
COMPONENT AHBMaster1
PORT (
   ahbmsttoarb : IN     ahb_mst_in_type ;
   clk         : IN     std_logic ;
   reset_n     : IN     std_logic ;
   ahbarbtomst : OUT    ahb_mst_out_type ;
   reg_addr    : IN     std_logic_vector (31 DOWNTO 0);
   reg_data    : IN     std_logic_vector (31 DOWNTO 0)
);
END COMPONENT;
COMPONENT adaptateur
PORT (
   ahbmstin  : IN     ahb_mst_in_type ;
   ahbslvin  : IN     ahb_mst_out_type ;
   clk       : IN     std_logic ;
   reset_n   : IN     std_logic ;
   ahbmstout : OUT    ahb_mst_out_type ;
   ahbslvout : OUT    ahb_mst_in_type ;
   clkbus    : IN     std_logic
);
END COMPONENT;
```

```
COMPONENT clocker
PORT (
   clk      : OUT    std_logic ;
   clkl     : OUT    std_logic ;
   clkv     : OUT    std_logic ;
   reset_n : OUT    std_logic
);
END COMPONENT;
COMPONENT mem_1r_1w
GENERIC (
   addr_width : INTEGER range 0 to 1000;
   word_width : INTEGER range 0 to 64 := 32;
   mem_depth  : INTEGER range 0 to 20000
);
PORT (
   -- The clock and the reset
   sys_clk    : IN     STD_LOGIC ;
   -- 3 read ports
   mem_rd_n   : IN     STD_LOGIC ;
   read_addr  : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
   -- write port
   mem_wr_n   : IN     STD_LOGIC ;
   write_addr : IN     STD_LOGIC_VECTOR (addr_width-1 DOWNTO 0);
   write_word : IN     STD_LOGIC_VECTOR (word_width-1 DOWNTO 0);
   -- the output word depending on input addresses
   read_word  : OUT    STD_LOGIC_VECTOR (word_width-1 DOWNTO 0)
);
END COMPONENT;
COMPONENT ptslv
PORT (
   hclk        : IN     std_logic ;
   hreset_n    : IN     std_logic ;
   readdata    : IN     std_logic_vector (31 DOWNTO 0);
   clk         : OUT    std_logic ;
   rd_n        : OUT    std_logic ;
   wr_n        : OUT    std_logic ;
   writedata   : OUT    std_logic_vector (31 DOWNTO 0);
   addr        : OUT    std_logic_vector (13 DOWNTO 0);
   ahbslavein  : IN     AHB_Slv_In_Type ;
   ahbslaveout : OUT    AHB_Slv_Out_Type
);
END COMPONENT;
COMPONENT registre
GENERIC (
   masters : integer := 8
);
PORT (
   reg_addr : OUT    registre_vector (0 TO MASTERS-1);
   reg_data : OUT    registre_vector (0 TO MASTERS-1)
);
END COMPONENT;

-- Optional embedded configurations
-- pragma synthesis_off
```

```
   FOR ALL : AHBArbiterDecoder USE ENTITY dynamic.AHBArbiterDecoder;
   FOR ALL : AHBMaster1 USE ENTITY dynamic.AHBMaster1;
   FOR ALL : adaptateur USE ENTITY dynamic.adaptateur;
   FOR ALL : clocker USE ENTITY dynamic.clocker;
   FOR ALL : mem_1r_1w USE ENTITY dynamic.mem_1r_1w;
   FOR ALL : ptslv USE ENTITY dynamic.ptslv;
   FOR ALL : registre USE ENTITY dynamic.registre;
   -- pragma synthesis_on


BEGIN
   -- Instance port mappings.
   I0 : AHBArbiterDecoder
      GENERIC MAP (
         MASTERS => MASTERS,
         SLAVES  => SLAVES
      )
      PORT MAP (
         hclk        => clk,
         hresetn     => reset_n,
         ahbmsttoarb => ahbmsttoarb,
         ahbarbtomst => ahbarbtomst,
         ahbslvtoarb => ahbslvtoarb,
         ahbarbtoslv => ahbarbtoslv
      );
   I1 : clocker
      PORT MAP (
         clk     => clk,
         clkl    => clkl,
         clkv    => clkv,
         reset_n => reset_n
      );
   I2 : mem_1r_1w
      GENERIC MAP (
         addr_width => 14,
         word_width => 32,
         mem_depth  => 3000
      )
      PORT MAP (
         sys_clk    => clk1,
         mem_rd_n   => rd_n,
         read_addr  => addr,
         mem_wr_n   => wr_n,
         write_addr => addr,
         write_word => write_word,
         read_word  => read_word
      );
   I4 : ptslv
      PORT MAP (
         hclk        => clk,
         hreset_n    => reset_n,
         readdata    => read_word,
         clk         => clk1,
         rd_n        => rd_n,
```

```
      wr_n        => wr_n,
      writedata   => write_word,
      addr        => addr,
      ahbslavein  => ahbarbtoslv(0),
      ahbslaveout => ahbslvtoarb(0)
   );
I3 : registre
   GENERIC MAP (
      masters => masters
   )
   PORT MAP (
      reg_addr => reg_addr,
      reg_data => reg_data
   );

g0: FOR i IN 0 TO CELLULE1 GENERATE
   I6 : adaptateur
      PORT MAP (
         ahbmstin  => ahbarbtomst(i),
         ahbslvin  => ahbarbtomst1(i),
         clk       => clkv,
         reset_n   => reset_n,
         ahbmstout => ahbmsttoarb(i),
         ahbslvout => ahbmsttoarb1(i),
         clkbus    => clk
      );
   I7 : AHBMaster1
      PORT MAP (
         ahbmsttoarb => ahbmsttoarb1(i),
         clk         => clkv,
         reset_n     => reset_n,
         ahbarbtomst => ahbarbtomst1(i),
         reg_addr    => reg_addr(i),
         reg_data    => reg_data(i)
      );
END GENERATE g0;

g1: FOR i IN CELLULE2 TO CELLULE3 GENERATE
   I8 : AHBMaster1
      PORT MAP (
         ahbmsttoarb => ahbmsttoarb1(i),
         clk         => clkl,
         reset_n     => reset_n,
         ahbarbtomst => ahbarbtomst1(i),
         reg_addr    => reg_addr(i),
         reg_data    => reg_data(i)
      );
   I9 : adaptateur
      PORT MAP (
         ahbmstin  => ahbarbtomst(i),
         ahbslvin  => ahbarbtomst1(i),
         clk       => clkl,
         reset_n   => reset_n,
         ahbmstout => ahbmsttoarb(i),
```

```
            ahbslvout => ahbmsttoarb1(i),
            clkbus    => clk
        );
    END GENERATE g1;

END struct;
```

```
--Modèle de clock clocker_behav.vhd(2)
-- hds header_start
--
-- VHDL Entity dynamic.clocker.symbol
--
-- Created:
--          by - udubois.etudiant (papineau)
--          at - 09:05:56 08/14/03
--
-- Generated by Mentor Graphics' HDL Designer(TM) 2001.0 (Build 178)
--
-- hds header_end
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY clocker IS
   PORT(
      clk     : OUT    std_logic;
      clkl    : OUT    std_logic;
      clkv    : OUT    std_logic;
      reset_n : OUT    std_logic
   );

-- Declarations

END clocker ;

-- hds interface_end
--
-- VHDL Architecture
--
-- Created:
--          by - udubois.etudiant (papineau)
--          at - 12:12:16 11/22/01
--
-- Generated by Mentor Graphics' Renoir(TM) 2000.4 (Build 5)
--
architecture behav of clocker is


    SIGNAL clkl_d : std_logic := '0';
     SIGNAL clkv_d : std_logic := '0';
    SIGNAL clk_d : std_logic := '0';

BEGIN
    reset : PROCESS
    BEGIN
        reset_n <= '0', '1' after 170 ns;
        WAIT;
    END PROCESS;

    clkl<= clkl_d;

    clk<=clk_d;
```

```
clkv<= clkv_d;

    PROCESS
          VARIABLE i   : integer := 0;
      BEGIN  -- PROCESS

          IF (i = 10000000) THEN
              i := 0;
          END IF;

          IF ((i MOD CELLULE1)   = 0) THEN
              clkl_d <= NOT clkl_d;
          END IF;

          IF ((i MOD CELLULE2)   = 0) THEN
              clkv_d <= NOT clkv_d;
          END IF;


          clk_d <= NOT clk_d;
          i := i+1;
          WAIT FOR  3125 ps;

    END PROCESS;



end behav;
```